

Modeling, Simulation, and Realization of Cognitive Technical Systems

Von der Fakultät für Ingenieurwissenschaften, Abteilung Maschinenbau und Verfahrenstechnik der
Universität Duisburg-Essen
zur Erlangung des akademischen Grades

eines
Doktors der Ingenieurwissenschaften
Dr.-Ing.

genehmigte Dissertation

von

Dennis Gamrad

aus

Voerde (Niederrhein)

Gutachter: Univ.-Prof. Dr.-Ing. Dirk Söffker
Prof. Dr.-Ing. Dr. h.c. mult. Eckehard Schnieder
Tag der mündlichen Prüfung: 18. November 2011

für

Anna, Helga, Lisa und Klaus

Danksagung

Diese Arbeit ist überwiegend während meiner Zeit als wissenschaftlicher Mitarbeiter am Lehrstuhl Steuerung, Regelung und Systemdynamik (SRS) der Universität Duisburg-Essen entstanden.

Mein aufrichtiger Dank gebührt in erster Linie Herrn Univ.-Prof. Dr.-Ing. Dirk Söffker für die Betreuung dieser Arbeit. Er hat diese Arbeit erst ermöglicht, mich unterstützt und zentrale Ideen beigetragen. Für die Ausbildung als Wissenschaftler hätte ich mir keinen besseren Lehrer wünschen können.

Ferner bedanke ich mich bei Herrn Prof. Dr.-Ing. Dr. h.c. mult. Eckehard Schnieder für das Interesse an dieser Arbeit sowie für seine Bereitschaft, das Zweitgutachten anzufertigen.

Während meiner Zeit als wissenschaftlicher Mitarbeiter hatte ich das Glück, in einem interkulturellen Umfeld tätig zu sein. Ich erinnere mich gern an die freundschaftliche Atmosphäre sowie zahlreiche fachliche und nichtfachliche Gespräche. Hierfür danke ich Hammoud Al-Joumaa, Lou'i Al-Shrouf, Dorra Baccar, Marc ter Beek, Kai-Uwe Dettmann, Philipp Ertle, Gregor Flesch, Xingguang Fu, Andreas Hasselberg, Frank Heidtmann, Marco Junglas, Amir Kazeminia, Friederike Kögler, Marcel Langer, Yan Liu, Matthias Marx, Mahmud-Sami Saadawia, Doris Schleithoff, Kurt Thelen, Yvonne Vengels, Chunsheng Wei, Heinz-Dieter Wend, und Fan Zhang. Darüber hinaus danke ich Krischan Wolters, Elmar Ahle und Hendrik Oberheid, von denen ich vor und während meiner Zeit als Doktorand viel gelernt habe.

Besonders danke ich meinen Eltern Helga und Klaus für die langjährige Unterstützung, Motivation und Liebe. Sie haben den Grundstein meiner Ausbildung gelegt und mich stets hilfreich beraten.

Abschließend aber nicht zuletzt gilt mein besonderer Dank meiner geliebten Ehefrau Anna, die meinen Blick für die wesentlichen Dinge bewahrt und mich in schweren Zeiten gestützt hat. Ihre Liebe erfüllt mein Leben.

Voerde (Niederrhein), November 2011

Dennis Gamrad

Abstract

This thesis presents a novel approach for the modeling, simulation, and realization of Cognitive Technical Systems.

In contrast to other approaches, in this thesis, the structure and dynamic of the real world is initially formalized by means of an intermediate level instead of implementing a technical model directly. Furthermore, human cognition is investigated in an integrated manner and based on experiments with a mobile robot, as an example for a complex technical system.

The formal description of human interaction and cognition is realized by Situation-Operator-Modeling (SOM), which can be implemented technically by patterns of high-level Petri Nets. With the state space of a SOM-based Petri Net, Human-Machine-Interaction can be analyzed, e.g., in order to detect human errors automatically. Furthermore, several cognitive functions, like planning, execution, perception, and learning, can be simulated. The different cognitive functions and related representations, which are all based on the same methodical background, are combined within an integrated cognitive architecture. Only the interplay among several functions and a novel kind of knowledge structuring, which contributes significantly to reduce the complexity of the real world, enable the realization of human-like behavior for technical systems. The system's capability to establish and to refine goal-directed behavior from interaction with the environment, also if no system-specific initial knowledge is available, is demonstrated by experiments with a mobile robot interacting within a dynamic office environment.

An additional value of this thesis for further research is especially given by the proposed generic approach for modeling, simulation, and analysis of Human-Machine-Interaction. Moreover, the formal description and implementation of the cognitive functions, the developed knowledge structuring, and the cognitive architecture may be applied to arbitrary kind of technical systems.

Kurzfassung

In dieser Arbeit wird ein neuartiger Ansatz zur Modellbildung, Simulation und Realisierung von Kognitiven Technischen Systemen präsentiert.

Gegenüber bestehenden Ansätzen setzt sich diese Arbeit insbesondere dadurch ab, dass die Struktur und Dynamik der realen Welt zuerst über eine methodische Zwischenebene formal beschrieben und erst danach technisch implementiert wird. Zudem wird menschliche Kognition ganzheitlich untersucht und direkt mit Hilfe von Experimenten mit einem mobilen Roboter, als Beispiel für ein komplexes technisches System, erprobt und entwickelt.

Die formale Beschreibung von menschlicher Interaktion und Kognition erfolgt über Situations-Operator-Modellbildung (SOM), welche über spezielle Muster höherer Petrinetze technisch implementiert werden kann. Durch den Zustandsraum eines SOM-basierten Petrinetzes ist es möglich, Mensch-Maschine-Interaktion zu analysieren, um beispielsweise menschliche Fehler automatisiert zu erfassen. Zudem können verschiedene kognitive Funktionen, wie Planen, Handeln, Wahrnehmung und Lernen simuliert werden. Die verschiedenen kognitiven Funktionen und entsprechenden Repräsentationen, welche auf der gleichen methodischen Grundlage basieren, werden in einer kognitiven Architektur zusammengeführt. Erst das Zusammenspiel verschiedener Funktionen und ein neuartiger Ansatz zur Wissensstrukturierung, wodurch insbesondere die Komplexität der realen Welt reduziert wird, ermöglicht die Realisierung menschenähnlichen Verhaltens für technische Systeme. Durch Experimente mit einem mobilen Roboter, der in einer dynamischen Büroumgebung interagiert, kann gezeigt werden, dass das vorgestellte System ohne anwendungsspezifisches Vorwissen in der Lage ist, zielführendes Verhalten aus der Interaktion mit der Umgebung zu erhalten und zu verbessern.

Ein Mehrwert aus dieser Arbeit für weiterführende Forschungsarbeiten ergibt sich insbesondere durch den vorgestellten generischen Ansatz zur Modellbildung, Simulation und Analyse von Mensch-Maschine-Interaktion. Zudem können die formale Beschreibung und die Implementierung der kognitiven Funktionen, der entwickelten Wissensstrukturierung und der darauf aufbauenden kognitiven Architektur auf beliebige technische Systeme übertragen werden.

Contents

1. Introduction	1
1.1. Motivation	1
1.2. Aims of this work	4
1.3. Organization of this work	5
2. Cognitive Technical Systems	7
2.1. Basics from Cognitive Psychology and Neuroscience	8
2.2. Related issues from Artificial Intelligence	10
2.2.1. Knowledge Representation and Reasoning	11
2.2.2. Machine Learning	16
2.3. Cognitive architectures	21
2.3.1. What is a cognitive architecture?	21
2.3.2. Overview about different systems	24
2.3.3. Critique	29
2.4. Existing technical applications and demonstrators	29
2.4.1. Humanoid robot ISAC	30
2.4.2. ACT-R and technical systems	31
2.4.3. Ripley the robot	33
2.4.4. SOM-based cognitive architecture	34
2.5. Summary and conclusion	37
3. Modeling of human interaction	40
3.1. Situation-Operator-Modeling (SOM)	41
3.1.1. Formalization of human errors	42
3.1.2. Hierarchical relations among situations	44
3.1.3. Action spaces	45
3.2. Simulation of SOM	47
3.2.1. Implementation with high-level Petri Nets	48
3.2.2. Example for a SOM-based Petri Net	48
3.3. State space analysis	50
3.4. Example: Automated detection of human errors	51
3.5. Concluding remarks	54
4. SOM-based design of Cognitive Technical Systems	55
4.1. Representational level	57
4.1.1. Different types of situations	57
4.1.2. Experiences and action spaces	59
4.1.3. Passive and active operators	61

4.1.4. Uncertain knowledge and conflicts	63
4.2. Knowledge and cognitive functions	65
4.2.1. Knowledge representation	65
4.2.2. Goal generation and selection	70
4.2.3. Planning	71
4.2.4. Perception	73
4.3. Key feature: learning	75
4.3.1. Operator's function and assumptions	76
4.3.2. Weighting of operators	77
4.3.3. Meta operators	78
4.3.4. Short-term memory	79
4.3.5. Situation's structure	80
4.4. The cognitive architecture ILCA	89
4.4.1. Structure of the representational level	89
4.4.2. Structure of the cognitive functions	91
4.4.3. Main cognitive cycle	92
4.4.4. Secondary processes	96
5. Realization of Cognitive Technical Systems	97
5.1. Specifications of technical systems to be controlled	97
5.2. Realization of a cognitive mobile robot	100
5.2.1. Hardware	100
5.2.2. Software	102
5.2.3. Example scenario: office navigation and service	103
6. Experiments and results	111
6.1. Generalization by new relations	113
6.2. Application of meta operators	121
6.3. Complexity reduction and hierarchical planning	123
6.4. Consideration of new facts from interaction	126
6.5. Refinement of perceptual capabilities	130
6.6. Methodical conclusions	134
7. Summary and future work	136
7.1. Summary	136
7.2. Future work	138
A. Implementation	141
Literature	155

List of figures

1.1. Scenarios with demand for autonomous systems	2
1.2. Examples for autonomous systems in environments of humans	3
2.1. Cognitive Technical Systems and related subjects	8
2.2. Principal fissures and lobes of the cerebrum viewed laterally [Gra18] . . .	9
2.3. Structure of an autonomous agent	11
2.4. Overview of knowledge representation models (ref. to [Hel06])	13
2.5. Example for a decision tree	18
2.6. Example for an Artificial Neural Network	19
2.7. Step ladder model (ref. to [Cac98])	23
2.8. Human Model of Reference (ref. to [Cac98])	24
2.9. Humanoid robot platform ISAC [KPB ⁺ 04]	30
2.10. Technical examples using ACT-R	32
2.11. Ripley the robot [Mav07]	33
2.12. Architecture of the cognitive autonomous system [Söf01a, Ahl07]	35
2.13. Different levels of learning [Ahl07]	36
3.1. From modeling to analysis of Human-Machine-Interaction	40
3.2. SOM-related notation and graphical representation	42
3.3. SOM-based formalization of the human error rigidity [Söf04b]	43
3.4. Hierarchical relations between situations	45
3.5. Hierarchization of action spaces	47
3.6. SOM-based net patterns	49
3.7. HMI-Analysis-Architecture	51
3.8. Arcade game (see also [Art10])	52
3.9. State-space-based detection of the human error rigidity	53
4.1. Required intermediate level between real world and technical system . . .	56
4.2. Class Situation	58
4.3. Class Characteristic	58
4.4. Connections between the different types of situations	59
4.5. Class Experience	60
4.6. Structure of an operator net	61
4.7. Class Assumption	62
4.8. Class Conflict	64
4.9. Action space with alternatives	66
4.10. Relations among action spaces, action models, and meta operators	67
4.11. Graphical representation of a meta operator	68

4.12. Class <code>metaOperator</code>	69
4.13. Class <code>Goal</code>	70
4.14. Goal generation and goal selection	72
4.15. Different representations of an apple	73
4.16. Measured and derived characteristics in focused situation	74
4.17. Different learning functions	75
4.18. Learning of operators' functions and assumptions	77
4.19. Robot learning the assumptions of the operator <code>o_{driveForward}</code>	81
4.20. Correlation of the general assumptions in the example scenario	83
4.21. Correlation of the characteristics required for conflict resolution	85
4.22. Sequence of different steps in the experiment	87
4.23. Average success of automatically learned relations in the example scenario	88
4.24. Memory-oriented connection between representation and functions	90
4.25. Three-level-structure of the cognitive architecture's modules	91
5.1. Cognitive architecture for arbitrary kinds of technical systems	98
5.2. Architecture communicates with technical system(s) over middleware	99
5.3. Sensors and actuators of the mobile robot	101
5.4. Mobile robot navigating through the office environment	104
6.1. Graphical User Interface to configure and to control the experiments	112
6.2. Flow chart of the experiment	114
6.3. Example for a new relation learned from interaction	115
6.4. Several mental action spaces generated during a run of the experiment	117
6.5. Comparison of two mental action spaces regarding correctness	118
6.6. Number of correct plans and user-selected operators	119
6.7. Results regarding the phases between two reached goals	120
6.8. Results of the analysis regarding the minimum path to the goal	120
6.9. Action space from the first experiment with wrong paths	121
6.10. Probability of different events during the experiment	122
6.11. Graphical illustration of the system's meta action space	124
6.12. Graphical illustration of the system's local action space	125
6.13. Meta action space of an environment with two offices	128
6.14. Local action space related to the operator <code>o₂</code> of the meta action space	129
6.15. Local action space related to the operator <code>o₁₀</code> of the meta action space	130
6.16. Local action space with several conflicts	132
6.17. Local action space after conflict resolution	133
A.1. Class diagram with SOM-based data structures	143
A.2. Operator net (implemented with Renew)	145
A.3. Sequence of operator nets (implemented with Renew)	146
A.4. Examples for function nets (implemented with Renew)	147
A.5. Net pattern for the generation of action spaces (implemented with Renew)	149
A.6. SOM-based visualization with Graphviz and ZGRViewer	150

List of tables

5.1. Characteristics defined for the scenario	106
5.2. Assumptions and relevant changes of the defined basic operators	110

1. Introduction

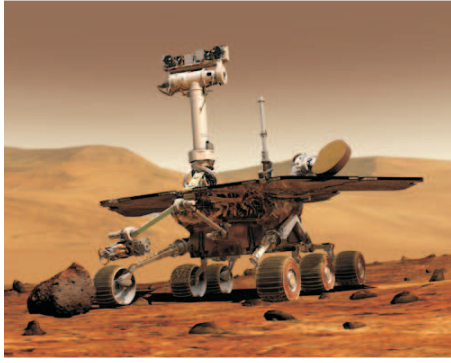
This thesis presents the conceptual development and implementation of a novel approach for the realization of autonomously performing Cognitive Technical Systems (CTSs). A commonly accepted assumption is that cognitive systems, which can also be technical, perform upon a mental representation of the real world (e.g., see [SBF⁺96]). In this regard, autonomy is related to a system's dependence on its initial knowledge (see [RN03a]). Hence, it is necessary that a system is able to learn by the construction and modification of its mental representation from interaction [Mic86] in order to enhance its performance [Sim83]. However, the independence from a certain problem or application field can not be achieved by simple parameterization or the adaptation of domain-specific models. Hence, a sufficiently flexible and generic representation of the real world is required.

In the following, the demand for autonomous systems is exemplarily illustrated for several application fields. Accordingly, the required key technologies for the realization of autonomy and central state-of-the-art approaches are highlighted and discussed. After that, the previous work and aimed challenges of this thesis are focused. Finally, each chapter of this thesis is briefly summarized.

1.1. Motivation

Technical systems which are denoted as autonomous are most often mobile systems, like service robots, driverless cars, autonomous underwater vehicles, or unmanned aerial vehicles. Systems like that usually act upon one or several models of their environment to derive further information from their sensor measurements. Hence, they are typically designed and optimized for a certain kind of application. However, in this case, the learning of completely new facts and the adaptation to environments changing in an unexpected manner are not considered. Accordingly, the independence from initial knowledge is not given or strongly limited.

Autonomous systems may be helpful in different areas in order to assist humans or to take on tasks that can not be done by humans, e.g., due to safety reasons, etc. An overview about different application fields is given in the following. Afterwards, the realization of autonomous systems by cognition-inspired approaches is discussed. Here, the utilization of methods from Artificial Intelligence (AI) as well as the application of cognitive architectures is focused.



(a) Model of the Mars rover Spirit



(b) Emergency landing in the Hudson river

Figure 1.1.: Scenarios with demand for autonomous systems

Demand for autonomous systems

One of the most typical scenarios for autonomous systems is the interaction of mobile robots with contaminated, extreme, or dangerous environments like the deep sea or the outer space (see Fig. 1.1(a)). In these often unstructured and (partially) unknown areas, autonomy can be necessary due to limited or impossible remote control, which may be caused by large distances, EMC problems, etc. In order to improve the robot's behavior, learning mechanisms can be implemented to map new facts and relations from the real world to the system's internal model. Hence, the autonomous realization of goal-directed behavior can be enabled or conserved respectively. Here, Simultaneous Localization and Mapping (SLAM) is investigated in robotics [TBF05]. However, in addition to the building of navigation maps, the whole variety of a robot's in- and outputs should be taken into account.

Assistance systems are used in Human-Machine-Systems to support humans, like the driver of a vehicle or the operator in a control room. These systems often provide additional well-visualized information and can offer optimized behaviors or strategies to be performed. In this regard, models of human performance can be utilized within a technical system in order to act upon the current or expected behavior of a human. This could be further optimized if the assistance system is personalized to a certain human (which may also be considered as unknown and complex interaction partner) or if it adapts to certain circumstances. Furthermore, flexible and autonomous assistance systems could also be applied in safety critical systems. Although the standard tasks in safety critical systems, like aircrafts or nuclear power plants, are fully automated, the pilot or human operator is remained in the loop as a fallback solution. Thus, the human, which is characterized by a high degree of flexibility, can overtake the control in the case of unexpected critical situations (see Fig. 1.1(b)¹). In such cases, assistance systems would be valuable if they could 'understand' new and unexpected circumstances.

¹Photo by the U.S. Army Corps of Engineers



(a) Driverless car Boss



(b) Vacuum-cleaner Roomba

Figure 1.2.: Examples for autonomous systems in environments of humans

More and more robotic systems are integrated in the environments of humans, like office buildings, private households, road traffic, etc. In this regard, the driverless car Boss (see Fig. 1.2(a)) is a famous example. The system is developed by the Tartan Racing Team from Carnegie Mellon University, which won the DARPA Urban Challenge in 2007 [UAB⁺08]. Moreover, several applications are already commercially available as the vacuum-cleaner Roomba [Jon06, TD07] cleaning rooms without user control (see Fig. 1.2(b)) or Lego Mindstorms robots [Wal01] which can be programmed even by children. Nevertheless, environments of humans are dynamic and not structured in a simple manner. Furthermore, intentions of humans can usually not be identified easily by technical systems. Accordingly, also robotic systems, interacting in environments of humans or with humans in general, require autonomy to adapt themselves to unusual and changing situations.

Realization of autonomous behavior

According to the argumentation at the beginning of this chapter, learning capabilities are necessary to realize autonomous systems, which are independent from their initial knowledge. Thus, the required internal representation of the real world can be adapted to enable goal-directed behavior permanently. However, in order to realize that also for technical systems with several sensors and actuators, the real world's complexity has to be reduced and the related knowledge has to be structured in a suitable way. In addition, the interplay among different information processing functionalities has to be organized.

The inspiration for the realization of autonomous behavior is typically given by natural cognitive systems, like human beings or highly developed animals. Cognitive systems perform complex behaviors in the real world and they are able to represent, understand, and process complex sensations with surprising facility. In this regard, also technical systems can be cognitive if they perform based on an internal representation of the real world [SHKH03]. In literature, those systems are commonly denoted as Cognitive Tech-

nical Systems or Cognitive Robots. According to some selected definitions, Cognitive Technical Systems are able to replicate central properties of the human brain in order to shape interaction [RHRS07]. They are information processing systems acting with the physical world and have cognitive capabilities like perception, reasoning, learning, and planning. Furthermore, Cognitive Technical Systems are also able to handle unexpected situations and they have a certain kind of self-awareness [Bra02].

In order to model and simulate cognition, methods from Artificial Intelligence are typically combined within integrated cognitive architectures. Some of these architectures are developed for robotic systems and they are strongly focused on capabilities like motor control, image processing, and speech recognition [WKG⁺07, KPB⁺04, Bre01]. Other architectures are based on psychological theories and experiments [LNR87, AL98]. These architectures are originally developed to model human performance, but they are also applied to control technical systems. Furthermore, several other architectures focusing different aspects of cognition are currently under development (see [LLR08, VMS07]).

Existing cognitive architectures are structured according to the applied technical methods or regarding certain basic representations. However, in this respect, the formalization of human interaction and cognition in the sense of an intermediate level between real world and technical model is not emphasized sufficiently. In addition, more research is necessary regarding meta modeling approaches, the implementation to technical systems, and learning mechanisms in order to realize autonomously performing Cognitive Technical Systems.

1.2. Aims of this work

In order to formalize the complexity of the real world and its mental representation in a unified manner, meta modeling approaches may be helpful. The meta modeling approach applied in this thesis is Situation-Operator-Modeling (SOM) providing moreover an intermediate level between real world and technical model.

The SOM approach introduced by SÖFFKER [Söf01c] extends the situations of the classical situation calculus [McC63] by an internal structure. Hence, the action logic as well as the situation's structure can be described by one meta model with multi-hierarchical structure. By means of the SOM approach, the interaction of technical systems and human operators (or drivers, etc.) as well as the human's mental representation of the real world can be described in a formal and unified way. Furthermore, SOM was used to develop an architecture for cognitive autonomous systems (see [Ahl07]). Here, AHLE demonstrated the implementation of a SOM-based architecture controlling a real technical system for the first time.

Based on the achievements of previous work, this thesis presents a novel approach for the realization of Cognitive Technical Systems. Here, also Situation-Operator-Modeling is used to represent the real world's structure and dynamics, but the proposed knowledge structuring and related cognitive functions are completely revised. In this regard, the thesis addresses

- modeling, simulation, and analysis of Human-Machine-Interaction,
- a representational level for the flexible processing of knowledge,
- suitable mapping from real to mental world with complexity reduction,
- the realization and combination of several cognitive functions, and
- the application to arbitrary technical systems in real world environments.

Each of the listed points addresses a separate problem, but can not be solved independently from the others. Thus, the entire project represents a challenging task, which is characterized by a high degree of complexity and a wide range of so far insufficiently considered aspects.

1.3. Organization of this work

In **Chapter 2**, an overview about Cognitive Technical Systems and related scientific disciplines is given. After some fundamentals about Cognitive Psychology and Neuroscience, the most relevant subjects of Artificial Intelligence including Knowledge Representation, Reasoning, and Machine Learning are detailed. In order to model and simulate the whole variety of human cognition, certain methods from AI are combined to integrated cognitive architectures. Hence, several examples with different structures and fundamentals are briefly explained. Furthermore, four cognitive architectures controlling technical systems are selected and described in detail. Finally, a summary and a conclusion about the whole chapter are given. Especially, the conclusion highlights those issues that are focused by this thesis and usually neglected within other approaches.

One of the basic ideas behind this thesis is the usage of a meta modeling approach to formalize the real world's structure and dynamics as well as its representation in the human mind in a unified manner. Hence, the SOM approach which is used for this purpose is presented in **Chapter 3**. After the methodical fundamentals, the simulation of SOM through high-level Petri Nets is presented. From the SOM-based Petri Net, a state space can be generated in order to analyze the represented interactions in the real world, which is also explained in detail. Finally, all central aspects of this chapter are illustrated by a simulation example presenting the automated detection of human errors.

The main contribution of this thesis is the development of a SOM-based framework for the simulation of cognition. This framework consists of several models and cognitive

functions based on the same methodical background and is used to realize cognitive architectures. In **Chapter 4**, all concepts and implementations behind the developed framework, such as the implementation of situations and operators by high-level Petri nets and Java objects respectively, are presented. In this regard, also the design of basic models, their combination to a complex hierarchical model, and related cognitive functions are detailed. In particular, the developed learning mechanisms modifying all models from interaction are described and validated. Finally, the cognitive architecture ILCA is presented, which is an integrated model of cognition based on the developed SOM-based framework.

In **Chapter 5**, the realization of Cognitive Technical Systems based on the developed framework is presented. Therefore, some specifications for systems to be controlled are described. This is especially related to the sensomotoric control of actuators and pre-processing of sensor measurements. Due to the fact that it is intended to connect the ILCA architecture with different kinds of systems, a middleware is used. The middleware realizes a network communication and offers interfaces to different programming languages. In order to validate the developed framework and cognitive architecture respectively, a mobile robot is presented as a representative demonstrator. The robot is a Pioneer 3 DX which is extended with further hardware and software. After the description of the robot's hard- and software, an example scenario is illustrated. The robot interacts within a dynamical office environment and has to learn from interaction. Therefore, the robot's sensor measurements represented as characteristics and the robot's actions represented as operators are described.

The experiments with the mobile robot are presented in **Chapter 6**. It is shown how the complex interaction with the real world environment is internally structured and how plans are generated and executed. In particular, the system's learning mechanisms are focused. They comprehend generalization of knowledge, resolution of conflicts, and the learning of new facts from interaction. In this regard, several snapshots of the system's working memory as well as some quantitative results are presented.

Finally, in **Chapter 7**, a summary and an outlook are given. The summary highlights the main contributions of this thesis and the outlook discusses how the proposed approach could be applied and extended in order to solve other problems in the future.

2. Cognitive Technical Systems

The term **cognition** derived from the Latin word *cognoscere* and the Greek word *gignoskein* respectively is related to the human capabilities of knowing, perceiving, and thinking. In Psychology, cognitive approaches are focused on the investigation of an organism's internal state, in contrast to behavioristic approaches considering exclusively observable stimuli and responses (see [SBF⁺96]). One well known definition from [SHKH03] concludes that also technical systems can be cognitive. Accordingly, cognitive systems are characterized by the ability to represent system-relevant aspects of the environment internally, which is the basis for cognitive functions. This allows flexible behaviors and yields in the consequence to what is usually called intelligence. Another definition from [Bra02] describes **Cognitive Technical Systems** as technical systems that *know what they are doing*.

In addition to the definitions above, the term Cognitive Technical Systems (CTSs) may also be used to denote an **interdisciplinary field of research** aiming to develop technical systems whose behaviors are inspired by or correspond to the cognitive capabilities of humans. In this context, different methods from Artificial Intelligence are typically applied to represent knowledge and to realize learning and reasoning tasks. Hence, CTSs can be considered as an intersection of Engineering Science, Cognitive Psychology, Neurobiology, Computer Science, and Artificial Intelligence (see Fig. 2.1). Besides the term 'Cognitive Technical Systems', the term 'Cognitive Robotics' (CR) is also often used in literature. Both terms denote areas with similar aims, applied methods, and challenges, except that CR is focused on robots as the considered kind of technical system. The large relevance of CTSs in today's research becomes apparent through the large number of **public funded projects**. As representative examples, the cluster of excellence 'Cognition for Technical Systems'¹, the transregional collaborative research center 28 'Cognitive Automobiles'², and 'Cognitive Systems, Interaction, Robotics'³ as a challenge of the seventh framework programme of the European commission CORDIS can be mentioned.

The following sections deal with the mentioned fields related to CTSs in order to give an overview about the fundamentals of this thesis. After a short introduction to Cognitive Psychology and Neuroscience, related fields from Artificial Intelligence are presented. In this regard, Knowledge Representation, Reasoning, and Machine Learning are focused as the most relevant topics for this thesis. After this, an introduction to cognitive architectures and some examples for typical demonstrators are given. The chapter

¹see <http://www.cotesys.org> (retrieved on December 12th, 2010)

²see <http://www.kognimobil.org> (retrieved on December 12th, 2010)

³see <http://cordis.europa.eu/fp7/ict/cognition/> (retrieved on December 12th, 2010)

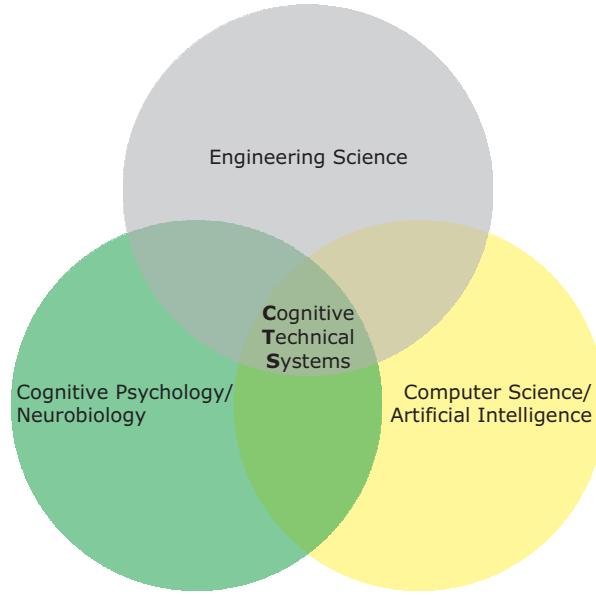


Figure 2.1.: Cognitive Technical Systems and related subjects

ends with a brief summary and a conclusion. The summary highlights central paradigms and research trends and the conclusion discusses those issues which are considered by this thesis for the first time.

2.1. Basics from Cognitive Psychology and Neuroscience

In **Cognitive Psychology**, it is investigated ‘how mind and psyche are organized’, ‘how intelligent thinking is produced’, and ‘how processes of thinking become visible in the brain’ [And07]. Typically, empirical studies are used to investigate issues as perception, attention, problem solving, language processing, or knowledge representation. Due to technological advances in recent years, Cognitive Psychology is more and more influenced by **Neuroscience** investigating the visualization of cognition within the brain [And07]. Therefore, ‘functional Magnetic Resonance Imaging’ (fMRI) is applied to visualize the used areas in the brain during thinking. It derives neural activities from the change of blood flow within the brain. Hence, it is also possible to describe those cognitive functions which are too complex for classical psychological methods.

In Fig. 2.2, the laterally view of the **human brain** is given in order to illustrate the locations of its functionalities which are inspiration for several issues in this thesis. The brain stem is one of the oldest parts of the human brain and is located in the lower part. It realizes basic reflexes and can also be found in a large number of animal species. The cerebellum is located in the lower right part and is partially responsible for the representation of well trained skills like bicycling or speaking. The youngest part of the human brain regarding human evolution is the cerebrum. The cerebrum is necessary for

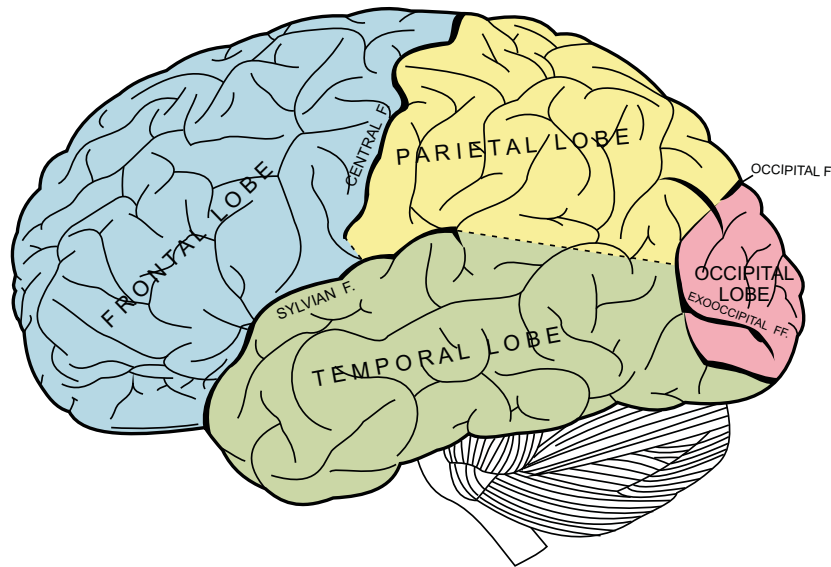


Figure 2.2.: Principal fissures and lobes of the cerebrum viewed laterally [Gra18]

higher cognitive functionalities and can be divided into primary and associative areas. The primary areas are related to information from a certain type of quality and the associative areas combine the information from several primary areas. As examples for primary areas, the visual areas are located in the back part on the occipital lobe and the auditory areas are located at the temporal lobe. Finally, the primary somatosensory cortex for the sense of touch and the primary motor cortex for the planning and execution of movements are located side by side at the central fissure.

As already mentioned, Cognitive Psychology and Neuroscience have **influence to other scientific disciplines** and vice versa. For example, Artificial Neural Networks which are inspired by biological neural networks within the central nervous system are used in different application to learn and model dynamical and non-linear system behaviors. This and other methods will be detailed in the following section. In addition, it is also investigated how humans are influenced by technological advances. In this context human factors in Human-Machine-Systems are usually investigated by heterogeneous teams of engineers and psychologists. Another example for a strong collaboration between Engineering Science, Neuroscience, and Medicine are ‘Brain-Machine-Interfaces’ (BMIs) which are used to exchange information directly with the brain, e.g., to develop artificial eyes [Dob00] or to control technical systems by thoughts [DNBH07].

2.2. Related issues from Artificial Intelligence

In order to describe Artificial Intelligence in a short and general manner, a relatively recent definition of one of its pioneers, John McCarthy, can be taken. McCarthy states that

it (Artificial Intelligence) is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable. [McC07]

In addition to that, [RN03a] summarizes and categorizes several different definitions. Accordingly, AI is related to human behavior, human functionality, rational behavior, or rational functionality. As the quotation of MCCARTHY, some of these short definitions include that intelligent machines (or agents, programs, etc.) are realized. However, these leads to a further question of ‘what is intelligence’. This again is also not easy to answer since the term has different meanings in different domains, which often contain several competing opinions. For instance, in order to measure intelligence or cognitive performance of humans, several tests were developed (see [Mac98]) to determine an **Intelligence Quotient (IQ)**. However, the existence of a unique measurable intelligence is not commonly accepted (e.g., see [Thu38]). As an example, in [Gar83], multiple kinds of intelligence are assumed. According to that, linguistic, logical-mathematical, musical, spatial, bodily-kinesthetic, interpersonal, and intrapersonal intelligence are distinguished.

Independently from the definitions, AI is a sub-domain of Computer Science and investigates human capabilities like perception, learning, reasoning, language, or problem solving. In the context of **autonomous systems**, which are focused in this thesis, the concept of an **agent** plays an important role. An agent

perceives its environment through sensors and acts upon that environment through actuators. The agent lacks autonomy if it relies on the prior knowledge of its designer rather than on its own percepts. A rational agent should be autonomous - it should learn what it can compensate for partial or incorrect prior knowledge. (see [RN03a])

Accordingly, an agent’s degree of autonomy is related to the agent’s dependence on its initial knowledge or in other words on the agent’s capability to extend and modify its knowledge through learning. Furthermore, functions for reasoning tasks are necessary to use the knowledge to inference about the real world and to predict their own and other agents’ behaviors (see Fig. 2.3).

Reasoning and learning are often investigated independently from each other which is also denoted as ‘**applied AI**’ [Ful06]. In this context, certain AI methods are developed to solve special problems. In contrast to that, the so-called ‘**strong AI**’ [Kur05] also

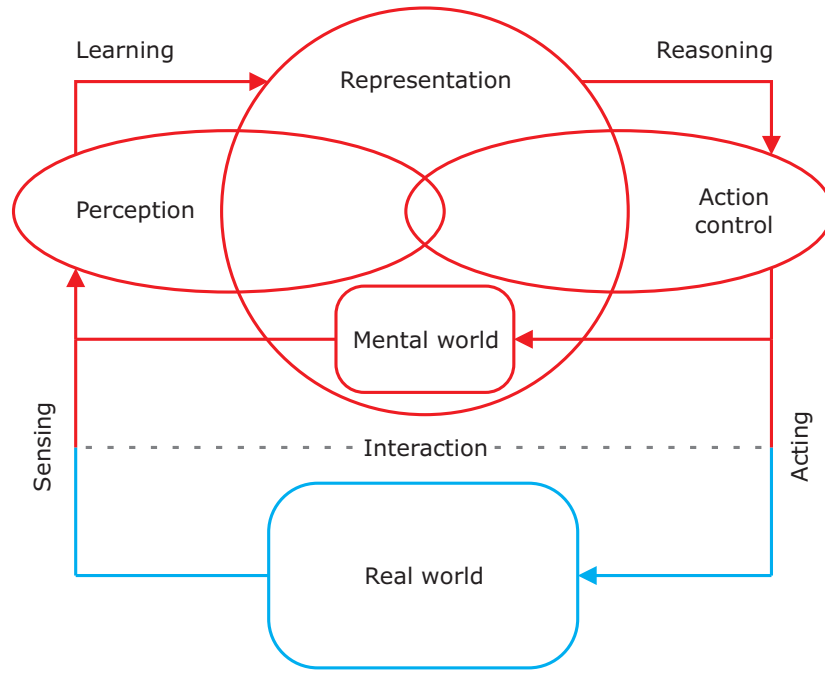


Figure 2.3.: Structure of an autonomous agent

denoted as Artificial General Intelligence [GP06] is related to the investigation of a whole intelligent system with several different capabilities combined in one architecture. These ‘cognitive architectures’ include AI methods for Knowledge Representation, Reasoning, and Machine Learning, which are described in the following sections. The fundamentals of cognitive architectures and some examples are presented afterwards in section 2.3.

2.2.1. Knowledge Representation and Reasoning

Since the cognitive revolution in the middle of the last century, **mental models** of the real world [JL83] are investigated by the arising field of Cognitive Science and related disciplines. Two of the main questions are ‘how those models should look like’ and ‘how they can be used to solve problems’. In this context, a large number of different approaches which can be directly implemented to technical systems are published in the fields of Computer Science and Artificial Intelligence. Hence, the definitions and methods given below are taken from these fields. According to [Sow00], **Knowledge Representation** can be defined as

a multidisciplinary subject that applies theories and techniques from three other fields: 1. Logic provides the formal structure and rules of inference. 2. Ontology defines the kinds of things that exist in the application domain. 3. Computation supports the applications that distinguish knowledge representation from pure philosophy. [Sow00]

Hence, a strong connection to Reasoning is described, which is moreover confirmed in [BL04]. Through **reasoning** the represented knowledge is used which is also described by the following definition from [RN03a].

The concepts that we discuss - the representation of knowledge and the reasoning processes that bring knowledge to life - are central for the entire field of artificial intelligence.

In order to represent knowledge, several different approaches with related reasoning mechanisms exist. They can be divided into

- symbolic systems where knowledge is represented explicitly, and
- subsymbolic systems where knowledge is represented implicitly.

In Fig. 2.4, an overview about different symbolic and subsymbolic representation methods is given. The different methods are divided into declarative (symbolic systems) and procedural (subsymbolic systems) knowledge representations. Declarative knowledge describes the ‘what’ as certain facts and relations about the real world and can be interpreted by humans. It is further divided into episodic knowledge which is related to special events and semantic knowledge which describes facts of the real world. In contrast to that, procedural knowledge describes the ‘how’ something works. Methods for the modeling and processing of subsymbolic represented knowledge are connectionist approaches where knowledge is described by the combination of several equal elements. Hence, these approaches, like Artificial Neural Networks, can not easily be interpreted by humans.

According to [SVS99], symbolic and subsymbolic approaches can be contained together in three different kinds of **hybrid intelligent systems**, which are detailed in the following:

- **Combined intelligent systems:** Symbolic systems use subsymbolic approaches and vice versa.
- **Transformational intelligent systems:** Symbolic and subsymbolic representations can be transformed into each other.
- **Coupled intelligent systems:** Symbolic and subsymbolic components communicate with each other.

In literature, also the term **ontology** appears in relation to the representation of knowledge. The term ontology is traditionally used in Philosophy and goes back to ARISTOTELE’s work about metaphysics (see [Ari04]). In Computer Science, it is defined as follows.

A specification of a representational vocabulary for a shared domain of discourse - definitions of classes, relations, functions, and other objects - is called an ontology. (see [Gru93])

Thus, an ontology can be considered as the description of a certain kind of knowledge representation. Nowadays, ontologies are investigated to define mediums for conversation of different knowledge-based systems. This is especially relevant for the ‘semantic web’ which should make the world-wide-web understandable for software [BLHL01].

Different kinds of methods for representation and reasoning

In the following, several different symbolic systems for the representation of knowledge are presented. As an example for a subsymbolic representation, Artificial Neural Networks are described in Section 2.2.2 since in this case the learning mechanism should be detailed, too.

Typical knowledge representations in AI are **formal languages** [RN03a] as ‘propositional logic’, ‘first-order logic’, ‘temporal logic’, or ‘fuzzy logic’ [Zad96]. They offer in contrast to higher programming languages as C++ or Java inference mechanisms to derive new facts from known facts. Propositional logic describes whether a fact is true or false. An extension of that is first-order-logic, which can also express objects, relations, and functions. It is more expressive than propositional logic and frequently used in AI (see [GRS03, RN03a]). Furthermore, in temporal logic, facts also hold at certain instances of time which are ordered. In all three mentioned formal languages the belief about a fact can be true, false, or unknown. In contrast to that, in probability theory

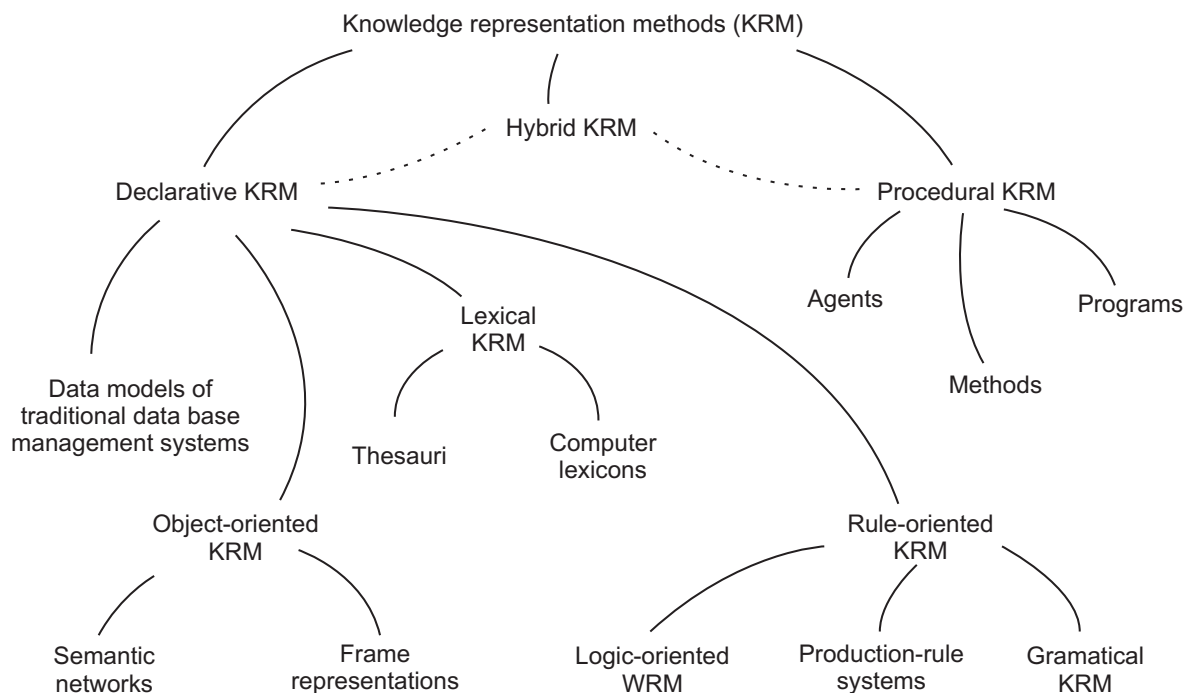


Figure 2.4.: Overview of knowledge representation models (ref. to [Hel06])

and fuzzy logic the belief about a fact has a certain degree of truth (e.g., between 0 and 1). In order to solve problems by formal languages, algorithms for forward and backward chaining (repeated application of inference rules) can be used. Forward chaining uses inference rules to extract new facts from known facts until a given goal is reached. In contrast to that, backward chaining starts from the goal and infers in the opposite direction.

Production systems are popular reasoning systems using forward chaining of first-order-logic. They consist of a database containing objects, a set of rules (termed productions), and a rule interpreter. Production rules are If-Then statements where the If-part describes the precondition and the Then-part describes the action to be executed. During runtime, the rule interpreter compares the content of the database with the If-part of the productions. If the precondition of more than one production matches, the rule interpreter has to use a mechanism for conflict resolution to select the production to be ‘fired’. If a production is chosen and fired, the database is changed and a new production to be executed has to be found. This circle of finding suitable productions, conflict resolution, and execution of productions’ actions repeats until no production can be applied anymore. Some of the most popular cognitive architectures as ACT-R, Soar, and EPIC are also production systems (see Section 2.3).

In order to represent changes within the real world, **Situation Calculus** originally introduced by McCarthy [McC63] can be used. This representation uses first-order logic to describe how situations are changed by actions [RN03b]. Furthermore, relations between objects within a situation can be described by so-called fluents. The Situation Calculus is often applied to robotics and can be used to infer a sequence of actions, which have to be executed to reach a given goal. A similar kind of discretization is also assumed by the SOM approach (see Section 3.1), which is used in this thesis for knowledge representation and meta-modeling as well. Here, a time-fixed, system, and problem equivalent situation [Söf04a] with a hybrid (different types of characteristics and relations) and open (hierarchical and adaptable) structure [Söf08] is used. Thus, SOM is suitable to describe the connections of planning, execution, learning, and errors within human interaction [Söf01c].

If reasoning is based on a representation of situations and actions, it can also be denoted as **planning**. According to [GNT04],

planning is the reasoning side of acting. It is an abstract, explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes.

Hence, planning is used to generate a sequence of actions from a certain problem description. The desired sequence of actions is termed plan and includes those actions which have to be executed to transfer a system from an initial to a desired state/situation. Hence, planning also enables a system to anticipate about future situations if the execution of own actions and/or actions of other agents or dynamical processes are taken into account. In technical systems, like mobile robots, it is distinguished between

local and global planning. Global or strategic planning usually uses static world models to generate a plan to be executed. The changes within the real world, which are not considered by the static world model, are handled by local or tactic planning. Here, a certain behavior is generated on a more detailed description of the current situation, e.g., to avoid collisions etc. In the case of robotic systems both planning strategies are usually applied in parallel. If the representation of the real world is very complex, the planning task can also be very time-consuming and should be simplified in a certain manner. A possible solution for this is hierarchical planning, where the planning task has several grades of abstraction [BL04].

If a problem may be described by a **state space representation**, which is defined as a set of discrete states and state transitions [Bec03] (in contrast to the state space representation in control), search algorithms can be applied to find a solution. A state space consists of a certain number of states and state transitions which transfer one state to another. Hence, state spaces can also be visualized as a directed graph where the nodes represent the states and the arcs represent the state transitions. Besides the state space, the initial and desired situations have to be defined to apply search algorithms. The search algorithms can be divided into complete (or blind) and heuristic methods. Complete methods as breadth-first-search or depth-first-search [GRS03] explore every node (if necessary) to find a solution. If more than one solution exists, usually the optimal one, e.g., the shortest path for not weighted state spaces, has to be chosen. However, if the states or state transitions are weighted, other approaches as the Dijkstra-algorithm [Dij59] can be applied to find a path with the minimum costs. Finally, heuristic methods as the A*-algorithm [HNR68] can be used to speed the search process up (e.g., for very large state spaces). However, in this case the optimal solution can not be guaranteed (see also [RSORS96]).

One of the oldest type of knowledge representation are **Semantic Networks (SNs)**. They are represented as directed or undirected graphs with concepts as nodes and relations as arcs. Some SNs can be very informal and are merely used for the visualization of knowledge. On the other side, they may also be highly formalized and can be used for automated inference. One of the simplest and oldest kind of a SN is termed ‘Definitional Network’ and uses a ‘is-a-relation’ to combine types of concepts with subtypes, e.g., a dog is a mammal and a mammal is an animal etc. Hence, a hierarchical structure results where the properties of the supertypes are copied to their subtypes. In [Sow92], six different kinds of SNs are distinguished and denoted as the already described Definitional Networks, ‘Assertional Networks’, ‘Implicational Networks’, ‘Executable Networks’, ‘Learning Networks’, and ‘Hybrid Networks’.

A further kind of structure for the representation of knowledge are **Frames**, which can be used to model stereotype situations or objects (see [Min74]). In general, a Frame consists of a name and a certain number of ‘slots’ (attributes of the Frame) with ‘fillers’ (values of the attributes). Furthermore, a frame can have one or several sub-frames taking the fillers of their super-frame as default fillers. However, the fillers of the sub-frames

can also be changed if the situation or object to be described differs to the described super-frame. For example, the super-frame `mobile robot` could have the slot `distance sensor` with the filler `ultra sonic` and the slot `color sensor` with the filler `camera`. Then, a sub-frame with the name `Robi the robot` could have the same slots and change the filler of the slot `distance sensor` to `laser range finder` and take the filler of the second slot from the super-frame as default.

2.2.2. Machine Learning

In the previous section, different kinds of knowledge representations and related reasoning methods are briefly presented. In order to close the circle of functionalities which are necessary for the development of autonomous agents (see Section 2.2), this section focuses on the acquisition of new knowledge and its modification, which is covered in AI by the field of Machine Learning. Two traditional definitions from Computer Science define Machine Learning as follows.

Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time. [Sim83]

Learning is constructing or modifying representations of what is being experienced. [Mic86]

Hence, a strong dependency to the improvement of the learning system's performance is assumed and the representation of knowledge is also taken into account. Nevertheless, the concrete challenges are also not detailed. In [Alp10], it is described that

machine learning is programming computers to optimize a performance criterion using example data or past experience... The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both.

Accordingly, Machine Learning is more related to the detection of certain patterns or regularities, which implies indirectly the improvement of the system's understanding and would provide more precisely information for problems to be solved. Furthermore, the representation is also considered since a model is built or modified.

Machine Learning has some connections to other areas including 'Data Mining' and 'Pattern Recognition'. The term **Data Mining**, which is an analogy to the 'mining' of coal or gold from sand and rocks, describes the application of Machine Learning methods on large amount of data to extract knowledge and patterns respectively (see [WF05]). Furthermore, it is one step in the process of 'Knowledge Discovery in Databases' (see [HK06]). Besides Engineering and Computer Science, it is also typically used in Economics (see [Alp00]). In contrast, **Pattern Recognition** is focused on the classification of objects into categories (see [TK08]) by the application of methods from Machine

Learning (see [Bis06]). As Knowledge Discovery in Databases, it describes a whole process and contains also other steps for the pre- and post-processing of data. Typical application scenarios can be found in Machine Vision, Character, or Speech Recognition.

The methods from Machine Learning and related areas are used in several different applications to solve certain kinds of problems. Depending on the kind of problem to be solved, the learning task can be assigned (see [RN03a]) to

- supervised learning,
- unsupervised learning, or
- reinforcement learning.

However, it has to be mentioned that a lot of methods are not restricted to one kind of learning. They offer a certain representation and learning mechanism and can be applied in different ways. Within **supervised learning** methods, the training examples to be learned contain in- and outputs. Hence, the result or classification of some input data is known during learning/training. In contrast to that, the training examples in **unsupervised learning** tasks do not include the output. Thus, new clusters have to be derived exclusively from the input data. Finally, **reinforcement learning** uses positive and negative rewards for executed actions based on a performance measure to learn an optimal strategy.

Another classification of learning methods is given [Lug01]. It is related to the kind of represented knowledge (see Section 2.2.1) and distinguishes symbol-based, connectionist, and emergent learning methods.

Overview about different learning methods

In the following, some of the most significant methods from Machine Learning as Decision Tree Learning, Artificial Neural Networks, Support Vector Machines, and Reinforcement Learning are briefly described. The chosen methods are relevant in some parts of this thesis and give an overview about different research directions.

Decision-Tree-Learning is one of the easiest, but very successful learning approaches. From a certain number of training examples (with several attributes) logical rules for sequential hierarchical decisions related to one certain question are derived. If the class of each training example is known (supervised learning), the approach can be used for the automated classification of other unclassified examples. The rules describe the general difference of the examples and can be visualized clearly as a tree. The tree consists always of one root, an arbitrary number of children nodes representing logic rules, and at least two leaves representing the answers to the rules. Besides the learning of examples with discrete values, the learning of continuous functions is also possible and

denoted as regression [RN03a]. In Fig. 2.5, an example decision tree describing ‘whether a robot can grip a green object’ is shown. The decision tree shows that the action ‘grip green object’ is only possible if a green object is detected, the distance to the object is shorter than 500 mm, and no other object is currently gripped.

The rules can be derived by recursive induction. Therefore, an attribute is chosen, which is sufficiently for a correct classification of the most examples. Then, it is tried to classify the remaining examples which can not be classified by the first attribute by another attribute and so on. The generation of decision trees can be realized by several different algorithms as the CART-algorithm (Classification And Regression Tree) [BFOS84], the CHAID-algorithm (Chi-square Automatic Interaction Detector) [SM64], the C4.5-algorithm [Qui93], and the ID3-algorithm [Qui86]. In order to increase the efficiency of the algorithms so-called ‘pruning methods’ can be applied. They are used to ignore some information during (pre-pruning) or after the learning phase (post-pruning). A typical problem of Decision-Tree-Learning is that the resulting rules are not correct or too simple if the training examples are not sufficiently different. Furthermore, the algorithm does not produce unambiguous results for training data from non-deterministic systems or if the training examples are partially erroneous (noisy). In this case, the decision supported by the most training examples might be preferred.

The training of **Artificial Neural Networks (ANNs)** is one of the most favored learning mechanism for the modeling of technical applications since they are also able to represent the behavior of non-linear dynamical systems (e.g., see [NRPH00]). ANNs are inspired by the net of nerve cells in the human brain and spinal cord. They can be classified as connectionist approach, where knowledge is represented by the connection of several simple elements (the artificial neurons) by weighted and directed links. Each of these elements contains an activation function (e.g., threshold, sigmoid function etc.), which defines the output based on the sum of all inputs and an individual threshold value. In general, an ANN consists of input neurons, output neurons, and hidden neurons. In a

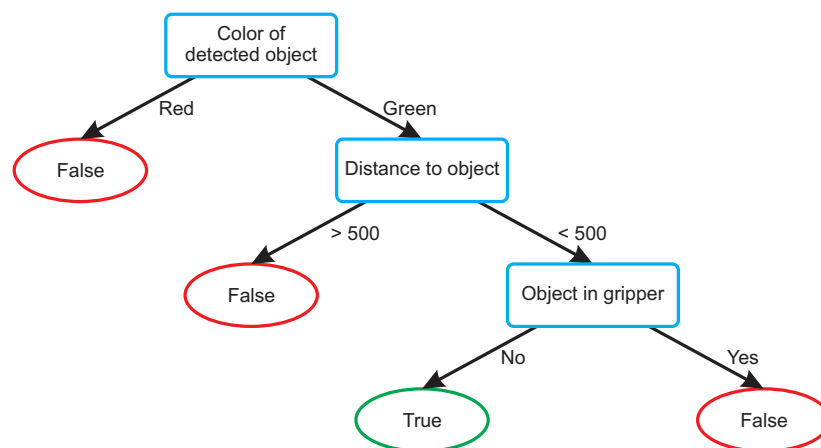


Figure 2.5.: Example for a decision tree

simple kind of ANNs, the feed-forward networks, the artificial neurons are only connected in one direction (from the input to the output) and are arranged in an input layer, an output layer, and one or several hidden layers. In Fig. 2.6, an example for a feed-forward network with three inputs, one output and four hidden neurons is illustrated. In addition to that, the hidden neurons of so-called feedback networks can be connected in both directions. During the training of an ANN, the number of artificial neurons, the threshold values, the activation functions, and the weights of their connections can be changed. However, usually the weights of the neuron's connections are changed. A typical algorithm for that is Backpropagation [RHW86], which determines the error of the network (difference between learned and original behavior) and modifies each weight in order to reduce this error. Although ANNs are very successful in many applications, there are also some drawbacks which have to be mentioned. Some weak points are that ANNs can not be interpreted by humans intuitively, they are relatively slow, and do not converge necessarily to a global optimum. In order to reduce some of these drawbacks, ANNs can be connected with other approaches like fuzzy logic etc. (see [NBFK03]).

If a huge amount of unstructured data has to be classified, **Support-Vector-Machines** (see [SS02, CST00]) are often the first choice to train a corresponding model. They have better generalization performance in comparison to ANNs and converge definitively to a global optimum. Within this approach, training objects are represented as vectors. These vectors are transferred to a space with a higher dimension, to find a hyperplane (for n -dimensional spaces, the hyperplane has the dimension of $n - 1$) which can separate the vectors. The vectors which are next to the hyperplane are taken as support vectors to define the hyperplane mathematically. In order to keep the effort of calculation small and the representation of the hyperplane as easy as possible, certain kernel-functions are used, which correspond to a certain high-dimensional space of features.

Reinforcement learning [SB98] can be applied to systems executing actions in environments which can be represented by a certain number of finite states. The method

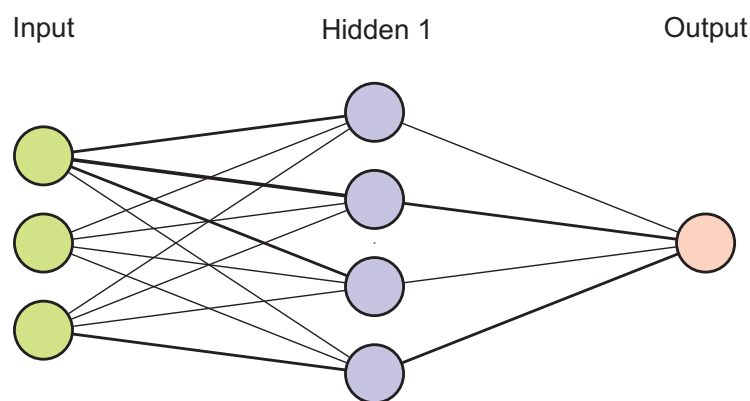


Figure 2.6.: Example for an Artificial Neural Network

aims to determine an optimal policy π describing which action should be performed in which state. This optimal policy directly leads to an optimal sequence of actions, which have to be performed to reach the related goal. If a deterministic world is assumed, the reinforcement problem can be defined by a set of finite states S , a set of finite actions A , a transition function $\delta(s_t, a_t)$, and a reward function $r(s_t, a_t)$. If an action a is executed in a state s_t , a new state s_{t+1} , and a related positive or negative reward result. In this regard, it is assumed that the resulting state and reward only depend on the current state and not on the interaction far in the past, which is also known as Markov Decision Process (MDP) [Bel57a]. In order to determine the optimal policy, also the rewards of following states/actions have to be taken into account. This can be achieved by ‘Dynamic Programming’ [Bel57b] using the results of several similar subproblems to solve one complex problem. The classical example for a Dynamic-Programming-based algorithm is ‘Value Iteration’. It calculates a utility for each states and represents them by a value function $V(s_t)$. In this case, a calculated utility depends on the reward of the related state as well as on the rewards of all successor states. Another very successful algorithm is Q-learning [Wat89, WD92]. It learns an action-value function $Q(s_t, a_t)$ from interaction and does not require a world model as the transition function.

Besides the learning methods mentioned above, several other successful methods exist. Some of them are listed in the following with representative references.

- **Genetic Algorithms** are used to solve optimization problems heuristically. They are inspired by natural evolution and use strategies as mutation, recombination, and selection to create new solution candidates from existing solutions for the given problem. (see [AWWB09, Gol89])
- **Case-based Reasoning** tries to solve a problem by known solutions of past problems which are similar to the problem to be solved. (see [Lea96])
- **Correlation and Regression Analysis** come from statistics and are used to detect relevant values from example data and derive general functions respectively. (e.g., see [Bob01])
- **Chunking** is the classical learning mechanism of the cognitive architecture Soar representing knowledge by production rules (permanent knowledge) and objects (temporary knowledge). Through chunking, a new production rule is generated from the solution of an impasse (lack of production rules) occurring during problem solving. (see [LRN86])

Several of the presented methods are included in commercial **software tools** as Matlab etc. In addition, some software tools containing the same algorithms are also available for free (e.g., under the GNU Public License). Two of these software tools are the ‘WEKA Data Mining Software’ [HFH⁺09] or ‘Rapidminer’ [MWK⁺06] which is applied within this thesis. It offers several so-called operators for pre- and post-processing of data as well as operators for different kinds of learning tasks. With an easy to use GUI,

sequences of operators can be created and executed. Furthermore, the resulting models and outputs can be visualized in different ways. Besides the GUI, the operators can also be used in own programs by implementing the corresponding Java library.

2.3. Cognitive architectures

The previous sections give an overview about some major topics and methods from Artificial Intelligence. However, modeling and simulation of functionalities comparable to human cognition requires more than the application of well-developed methods related to single functions as learning or planning. Moreover, if the structure of the human brain is considered, it can be supposed that only large-scale theories and complex systems of several functionalities are able to conquer this challenge. The related theories and software systems in this context are denoted as cognitive architectures.

In the following parts of this section, different definitions and approaches are summarized to give an impression of what a cognitive architecture is. Additionally, in Section 2.3.2, some well-known and successful architectures are classified and briefly described.

2.3.1. What is a cognitive architecture?

According to [New90], an integrated theory of mind is necessary for the modeling of human cognition to realize real cognitive systems. Additionally in [LLR08], the following classical AI-oriented definition is given.

A cognitive architecture specifies the underlying infrastructure for an intelligent system. Briefly, an architecture includes those aspects of a cognitive agent that are constant over time and across different application domains. These typically include: 1) the short-term and long-term memories stores the agent's beliefs, goals, and knowledge; 2) the representation of the memories content organization into larger-scale mental structures; and 3) the functional processes that operate on these structures for usage and learning

Hence, a cognitive architecture builds a fixed framework containing knowledge as the variable part. Unfortunately, it is sometimes (usually in AI) exclusively aimed to improve a special algorithm to a certain application [Lan06]. In contrast to that, a cognitive architecture covers the whole area of human cognition (see [ABB⁺04, LLR08]).

In [LLR08], several **capabilities of cognitive architectures** are summarized. It is stated that some capabilities as recognition and decision making are necessary as minimum requirements for a cognitive architecture. However, the more capabilities are supported, the more capable the architecture is to represent the whole range of human cognition. These capabilities involve recognition and categorization, decision making and choice, perception and situation assessment, prediction and monitoring, problem

solving and planning, reasoning and belief maintenance, execution and action, interaction and communication, and remembering, reflection, and learning.

In addition to the description above, simulation of cognition is also claimed by a large number of approaches which do not correspond to the classical definition. Accordingly in [VMS07], the term ‘cognitive system’ is considered more widely. It is stated that

...the minimal configuration of a system that is necessary for the system to exhibit cognitive capabilities and behaviors: the specification of the components in a cognitive system, their function, and their organization as a whole.

which emphasizes the need of systems that are developmental and emergent, rather than preconfigured. In order to distinguish different approaches, two paradigms of cognition are proposed for classification. The paradigms comprehend the **cognitivist approach** and the **emergent systems approach**. The cognitivist approach includes systems which process and represent information symbolically and the emergent systems approach includes systems which have the ability of self-organization and are further divided into connectionist systems, dynamical systems, and enactive systems. In addition to that classification, also hybrid systems are considered. They include characteristics of both, the cognitivist and the emergent systems approach.

In [Cac98], also two different perspectives for the study of cognition are described. On the one hand, it is investigated how cognition works in the human mind and on the other hand it is investigated how cognition (considered as the input/output-behavior of humans) is used to perform certain tasks. It is stated that

***micro-cognition** denotes the academic pursuit of modeling psychological phenomena and **macro-cognition** lies, instead, in an engineering context, where the primary purpose is to construct a representation of a phenomenon or system. This is then used to calculate or predict how the system will develop.*

Hence, micro-cognition investigates cognitive processes and mechanisms as the human memory, problem solving, or learning in detail. This does not necessarily have to be related to a certain application area. Examples for systems which can be related to micro-cognition are cognitive architectures with a background in cognitive psychology. In contrast to that, macro-cognition is related to systems modeling cognitive phenomena which are related to interaction with the environment. Here, it is focused how a task is performed and how efficiently it is in order to reach a goal.

The **step ladder model** of RASMUSSEN (see [Ras83, Ras86]) models the decision making process of human operators in complex physical systems (see Fig. 2.7). It assumes three different levels of cognitive behaviors as the knowledge-based, the rule-based, and the skill-based level [Ras79], which is also applied in this thesis. Furthermore, the step ladder model contains ‘states of knowledge’, which are linked by ‘information processing activities’. The application of known rules by using all or some information

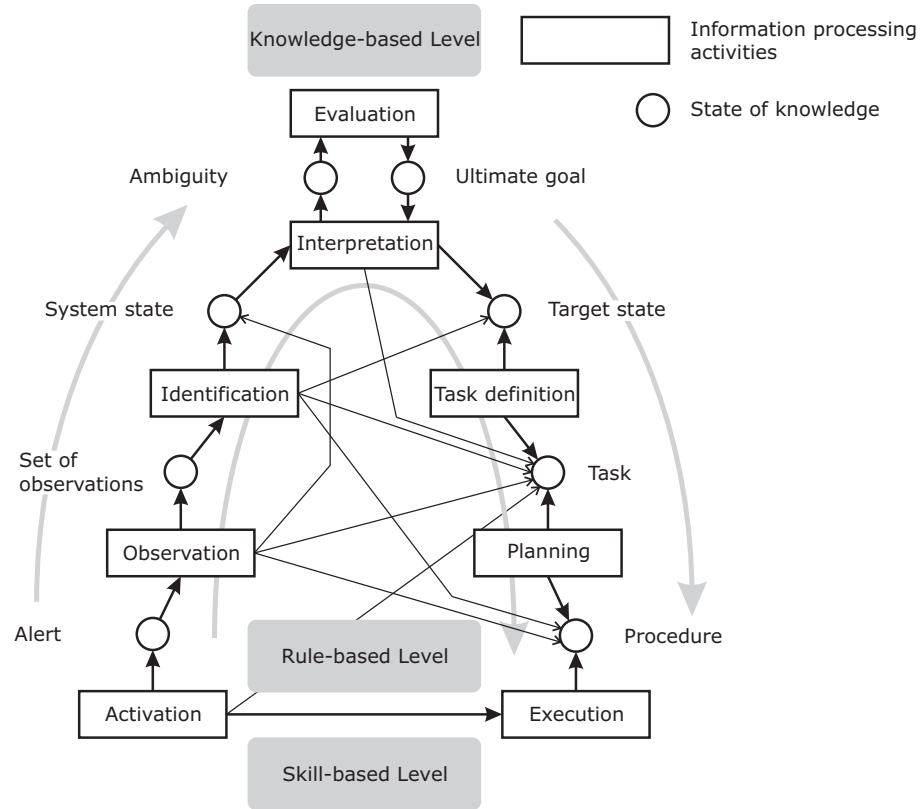


Figure 2.7.: Step ladder model (ref. to [Cac98])

processing activities to find a decision is denoted as rule-based behavior. Skill-based behavior occurs if the activities of activation and execution are directly linked. This is the case if a behavior is well experienced and no doubt about alternatives exists. However, if the human operator is confronted with unexpected or unknown situations and the available rules are not sufficiently to find a decision, knowledge-based behavior is necessary. Therefore, knowledge about the system and appropriate reasoning mechanisms have to be used.

In [Cac98], the **Human Model of Reference** is presented (see Fig. 2.8). It combines some fundamental aspects for the modeling and simulation of human cognition and is one of the major approaches that inspired some central ideas of this thesis. The model contains four main cognitive functions and two cognitive processes. The cognitive functions are perception, interpretation, planning, and execution, which are summarized under the term **PIPE**. The cognitive function 'perception' is related to the sensory input of a human and moreover is also influenced by the operator's expectations. Hence, perception is more than sensing as assumed by some other approaches. The perceived information is further processed by the function 'interpretation' relating this information to the knowledge base for the identification of relevant meanings. In the function 'planning', decisions are made to generate a plan of actions to be executed. This is real-

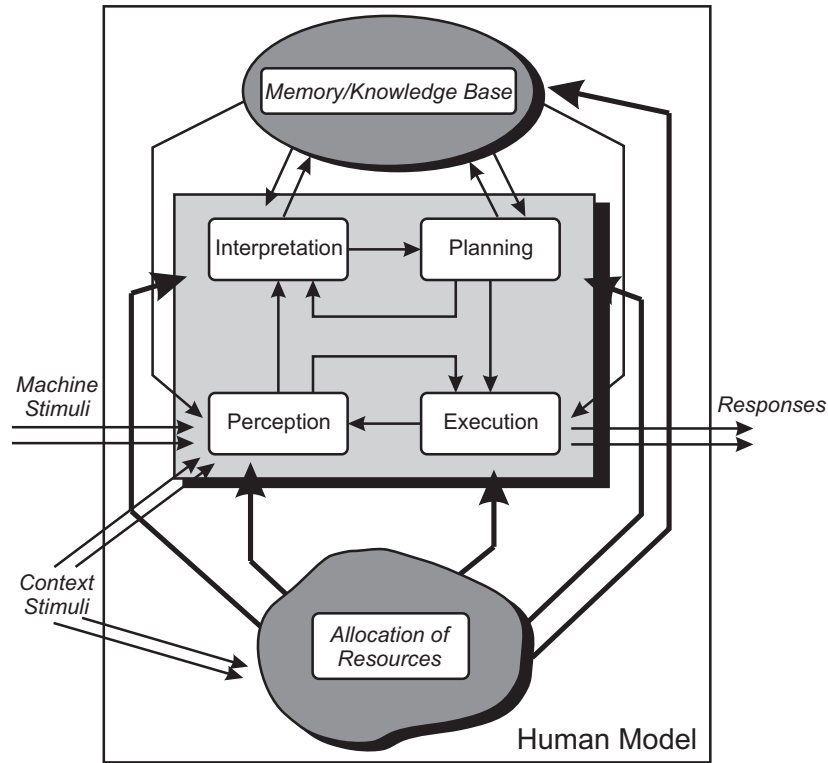


Figure 2.8.: Human Model of Reference (ref. to [Cac98])

ized by the function ‘execution’, which can result in performing an action or in starting a new cognitive process. The considered cognitive processes are ‘Memory/Knowledge Base’ (**KB**) and ‘Allocation of Resources’ (**AoR**). The Knowledge Base is connected to all cognitive functions and supports them with (learned) knowledge. Allocation of Resources influences also the whole system since it describes the level of operability and relative sequence of cognitive functions (see also [Wer06]). Furthermore, it defines how much and in which way knowledge is available.

2.3.2. Overview about different systems

The first architectures aiming simulation of intelligent behavior are the **Logic Theorist** [NS56] and the **General Problem Solver (GPS)** [NS61]. The GPS uses ‘means-ends analysis’ as search heuristic in order to solve logic problems, but is not applicable to arbitrary kinds of problems and does not include learning mechanisms [GRS03]. Nevertheless, it had large impact to the development of subsequent approaches.

The cognitive architectures presented in the following are classified as cognitivist, emergent, and hybrid approaches as proposed in [VMS07].

Emergent systems approaches

Although the majority of approaches presented below assume the existence of a representational level as mandatory aspect for the simulation of cognition, they are sometimes compared with purely reactive systems (see [CTN07, VMS07]). A popular example for such a system is BROOKS' **Subsumption Architecture** [Bro99], which neglects representation [Bro91] and emphasizes embodiment as important issue to realize intelligence. Nevertheless, also in this thesis the existence of a representational level is assumed as necessary condition in order to classify a system as cognitive.

According to [VMS07], the Subsumption Architecture and similar approaches based on autonomous agents and behaviors can be assigned to the class of emergent systems. Further approaches in the context of emergent systems are the **Global Workspace Cognitive Architecture** [Sha06] implemented as connectionistic system, **SDAL (Self-Directed Anticipative Learning)** [CH00] a dynamical embodied approach, and the **SASE (Self-Aware Self-Effecting)** [Wen04] architecture with many aspects of enactive approaches. Furthermore, aspects of enactive and connectionistic models can be found in the robot platforms called **Darwin** [KE08].

In the **Society of Mind** theory [Min86], it is assumed that intelligent behavior results from the interplay of several independent not intelligent agents. Hence, it is also a typical and popular example for a connectionist approach. Although the Society of Mind theory does not describe the technical realization of a cognitive architecture in detail, it inspired many other approaches and can be found in some of today's architectures. In a following book of MINSKY [Min06], the relevance of emotions is taken more into account in the way that they are considered as 'ways to think'. A first implementation of this Emotion Machine theory is the cognitive architecture **EM-ONE** [Sin05], which is demonstrated by controlling robots in a simulated world environment.

Cognitivist approaches

Classically, cognitive architectures are less related to the realization of autonomous systems, but rather to the exact modeling and simulation of human performance under psychological constraints. Three of the most cited cognitive architectures are

- **Soar (prev. State, Operator, And Result)** [LNR87],
- **ACT-R (Adaptive Control of Thought–Rational)** [AL98], and
- **EPIC (Executive-Process/Interactive Control)** [KM97].

All three approaches are implemented as production systems (see Section 2.2.1) and represent knowledge symbolically. Additionally, ACT-R also contains subsymbolic representation. Contrary to EPIC, the architectures Soar and ACT-R provide different kinds of learning mechanisms and are moreover available as freeware. Furthermore, in

addition to the originally aimed usage for human performance modeling, ACT-R and Soar are also applied to control technical systems. As an representative example, Section 2.4.2 presents the combination of ACT-R with different kinds of technical systems and provides more information about its knowledge representation and learning mechanisms.

The cognitive architecture **ADAPT (Adaptive Dynamics and Active Perception for Thought)** [BLL04] is based on the Soar architecture and especially designed to control robots. As denoted by its name, two of its main features are active perception and adaptive dynamics. ‘Active perception’ means that the system’s perception depends on the current goal and context. On the other side, the feature ‘adaptive dynamics’ generates complex behaviors from one or several simple behaviors. The architecture is implemented and tested on a Pioneer 2 mobile robot platform.

Another classical cognitivist approach is the architecture **ICARUS** [LC04, LC06]. The approach focuses on perception and action and is intended to be implemented on physical agent [LCR09] as long-term goal. It contains two kinds of knowledge divided into concepts for perception and skills for action. Furthermore, long-term and short-term memories are implemented for both kinds of knowledge. The long-term memory has a clear hierarchical structure and is related to the short-term memory containing special knowledge related to certain structures in the long-term memory.

Prodigy is a modular and engineering-oriented architecture developed originally for problem-solving and planning [CEG⁺90]. The knowledge is represented as a set of ‘operators’ which are related to physical actions and are described by conditions and effects. These operators build the problem space used to generate plans from initial to desired states. Furthermore, Prodigy provides several kinds of learning mechanisms [VCP⁺95]. The architecture is also implemented within a framework on a mobile robot [HV96].

The **CogAff (Cognition and Affect)** scheme [Slo01] describes a theoretical approach for various kinds of intelligent systems and is not focused on human intelligence in particular (a related scheme specified to humans is termed HCogAff). As known from other approaches, a framework with three different kinds of processing levels is assumed, however, a certain kind of representation is not specified and related computational simulation systems are not existing yet.

Within the **Psi** theory [Dör01], the behavior of a cognitive agent is influenced by drives, emotions, and motivations. The theory is implemented to control a software agent in a simulation environment [DSD01]. Here, the steam engine ‘Psi’ exists on a lost island and learns how to interact with objects and how to satisfy its drives. Another implementation of the Psi theory is microPsi [Bac09] which is also used as a robot control architecture [BBV06]. As an example, microPsi is implemented on a mobile robot which is based on a Mac Mini [Bac05].

Hybrid approaches

Besides the described emergent and cognivist approaches, also some hybrid architectures can be found in the literature. Accordingly, hybrid systems as Cerebus, LiCAI, Polyscheme, CLARION, LIDA, and an architecture for humanoid robots are briefly described in the following.

The **Cerebus** project [Hor01] combines parallel and distributed computations as known from behavior-based robotics with an approach for symbolic representation and inference. As a main difference to classical approaches, the proposed architecture does not contain one central unit for representation and inference. Instead of that, several different representations as a semantic network and an inference network are implemented. Hence, one certain object or concept can be described in different ways. The architecture is demonstrated with a mobile robot and communicates with humans over spoken language. However, the implementation of learning mechanisms has not been presented so far.

The model **LiCAI (Linked model of Comprehension-based Action planning and Instruction taking)** [KP97a] simulates the interaction between a human operator and an interface. In the first of two phases of execution, connectionist networks of alternative meanings of sentences or alternative actions are generated. Then, a selection process follows, which allows to simulate how tasks are performed by exploration. Extensions of LiCAI are **LiCAI+** [KP97b] incorporating learning mechanisms and **CoLiDeS** [KBP00] simulating navigation on the World Wide Web.

The **Polyscheme** framework [Cas02, CBB⁺10] is used to create an equally named cognitive architecture containing several modules (denoted as ‘specialists’) which are based on different kinds of representation and inference methods. It uses a ‘focus of attention’ approach to execute hybrids of algorithms. In the main control cycle, a ‘focus manager’ selects a certain proposition (relation which can be true or false) to be focused. All specialists take this proposition and give their opinion on its truth. Then, the specialists reason again about the proposition’s truth by taking the results of the other specialists into account. The final result of each specialist is then provided to the focus manager which selects the new proposition. The approach is implemented to different kinds of systems as a mobile robot [CTBS04] or a heterogeneous database retrieval system [Cas03].

The hybrid cognitive architecture **CLARION (Connectionist Learning with Adaptive Rule Induction ON-line)** [SZ06] consists of an ‘action-centered subsystem’, a ‘non-action-centered subsystem’, a ‘motivational subsystem’, and a ‘meta-cognition subsystem’. The action-centered as well as the non-action-centered subsystem represent procedural knowledge implicit on a bottom level and declarative knowledge explicit on a top level. In the case of the action-centered subsystem, symbolic rules are stored on the top level and neural networks are stored on the bottom level. The action-centered

subsystem controls the non-action-centered subsystem representing general knowledge about the world as chunks and associative rules on the top level and associative memory networks on the bottom level. However, the architecture can also perform without any initial knowledge. Therefore several bottom-up and top-down learning mechanisms are used. As an example for bottom-up learning, implicit knowledge on the bottom level (e.g., acquired by reinforcement learning) can be used to extract explicit rules stored on the top level (see [SMP01]). In order to reason about the knowledge, also several mechanisms as similarity-based and rule-based reasoning are applied. Finally, the motivational subsystem represents explicit goals and drive states (which may generate the goals) and the subsystem for meta-cognition regulating goal structures and other cognitive processes (see [Sun07]). CLARION, whose source code is freely available, is used to solve several different problems. However, the implementation to a technical system seems not to be focused.

Another hybrid system is the **LIDA** framework [FP06, RBDF06] extending the previous IDA [Fra03] system by several learning mechanisms. It contains different kinds of knowledge representations and is executed in a cognitive cycle. Furthermore, three different types of learning as ‘perceptual learning’, ‘episodic learning’, and ‘procedural learning’ are integrated. The architecture is intended to create intelligent (human-like) performing agents interacting in real or simulated environments. However, experiments with real technical systems are not presented so far.

In [BMS⁺05, Bur07], a **cognitive architecture for humanoid robots** is presented. It consists of a hierarchical perception system divided into three modules for low-level (fast interpretation, no memory access), mid-level (further recognition with memory access), and high-level (multimodal fusion, situation recognition, etc.) perception. On the opposite side, actions are planned, coordinated, and executed by a three-level hierarchical task handling system. In order to plan a sequence of actions, a global knowledge base acting as long-term memory is used. The knowledge base contains different kinds of representations which are partially related to the considered application. The scheduling of tasks and resource management is realized by an ‘execution supervisor’. All three levels of the task-handling and perception system are connected with so-called ‘active models’ acting as working memory of the system. The communication with humans is realized by a ‘dialogue manager’ serving as intermediate component between task planning and perception. As learning mechanism, tasks, flows of actions, and additional information as all occurring states etc. are simply stored. The cognitive architecture is developed for the humanoid robot ARMAR III and its functionalities are illustrated by a scenario where a robot arm interacts with a human mentor in order to learn and perform a puzzle game.

2.3.3. Critique

In [LLR08], an overview of open issues regarding cognitive architectures is given. It is distinguished between new or improved capabilities that are required and the structures and processes supporting these capabilities:

- Regarding new **capabilities for cognitive architectures**, it is written that more research on capabilities of categorization and understanding is necessary. Furthermore, cognitive architectures should also contain the representation of episodic knowledge which is not considered sufficiently. Moreover, a variety of representational schemes should be supported instead of only logic-based formalisms. The processing of natural language is rarely used for communication tasks and also emotions and their connections to other cognitive processes should be investigated.
- Although a lot of architectures are used to **control technical systems** as mobile robots etc., the physical embodiment and its influence to mental processes is not considered sufficiently. In this context, also the management of resources to selectively focus on the agent's perceptual attention, effectors, and current tasks has to be implemented. Finally, it is pointed out that the reuse of functional capacities as well as the reuse of knowledge plays an important role in engineering.
- Improvements regarding **cognitive architecture's structures and processes**, alternative representational frameworks instead of the often used production systems should be considered. Furthermore, the utilization of knowledge should be more dynamically and related on the current situation. Last but not least, learning should also be focused. Although a lot of different approaches were presented and some problems are already solved, the realization of a real autonomous system which can be compared to a human is not solved yet.

2.4. Existing technical applications and demonstrators

In addition to the cognitive architectures presented above, several selected examples for successful implementations to technical demonstrators are described more detailed in the following. As important criteria in the context of this thesis, systems are selected that

- interact with real environments,
- behave upon a certain approach, like a cognitive architecture, and
- should represent and acquire knowledge from interaction.

According to these criteria, three different systems are selected. Although other systems as those described in the previous section might also match to the given criteria, the following detailed description should be limited to only a few representative examples.

Furthermore, the first implementation of a cognitive architecture based on Situation-Operator-Modeling (SOM) is presented. Since the results and ideas of this work influenced the process of this thesis in particular, the previous concepts of SOM-related knowledge representation and learning mechanisms are described in detail.

2.4.1. Humanoid robot ISAC

The humanoid robot **ISAC (Intelligent SoftArm Control)** [KPB⁺04] is a research platform for service robotics and Human-Robot-Communication (see Fig. 2.9). It consists of two 6-degree-of-freedom robot arms, which are actuated by pneumatic muscles. Each robot arm is connected to an anthropomorphic manipulator with four touch-sensitive fingers and a force-torque sensor at the wrists. Furthermore, two controllable color cameras for visual sensing are mounted on the top of the robot.

In order to control ISAC, a multi-agent software system called **Intelligent Machine Architecture (IMA)** [PWK97] is used. It consists of different memory structures as well as different independent software agents interacting with each other to produce intelligent behavior. Here, atomic agents and compound agents consisting of several atomic or compound agents are distinguished. ‘Hardware agents’ accessing the sensors and actuators of the robot can store sensory data in the short-term memory realized by a ‘Sensory EgoSphere’ [PHKW01]. The Sensory EgoSphere represents a geodesic dome surrounding the robot. Hence, currently sensed as well as previous sensed information

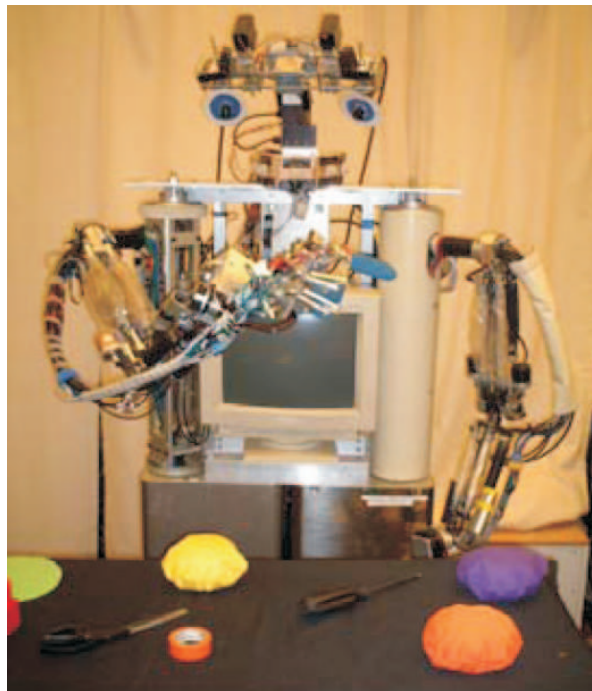


Figure 2.9.: Humanoid robot platform ISAC [KPB⁺04]

can be related to certain locations at the dome. Furthermore, a long-term memory stores behaviors and parameters (to execute the behaviors) as procedural knowledge and information about objects and percepts as semantic knowledge. Based on short-term and long-term-memory a working memory provides task-related chunks, which are managed by a neural network. All memories are connected with a ‘self agent’ which is responsible for cognitive activities as sensor signal monitoring, planning, and cognitive control. It is moreover connected to a ‘human agent’ monitoring the features of persons and deriving their intentions.

As learning mechanism, ISAC can be trained to identify simple objects in its environment. Furthermore, a method called ‘Verbs and Adverbs’ [JM03] is used to generate new motions (based on one or several parameters) from a set of example motions. In the case of ISAC, the behaviors **handshake**, **reach**, and **wave** are trained. In order to acquire a desired behavior, the artificial neural network in the system’s working memory can also be learned from interaction with the environment. Here, Reinforcement Learning is used to train which chunks should be loaded by the working memory to reach a certain goal [SNW⁺04, GH06]. Finally in [EFK⁺08], the training of a robot’s attention by the usage of ‘affordance relations’ connecting situational features to behaviors is presented.

The described functionalities of the IMA are demonstrated by different scenarios where ISAC interacts with humans and a table of colored objects. For example, ISAC communicates with persons by spoken language and can pick and place objects on the table. Here, the demonstration of the system’s learning mechanisms is focused in particular. Furthermore, the IMA and parts of it are also implemented to other kinds of systems, like mobile robots.

2.4.2. ACT-R and technical systems

ACT-R is one of the most popular and widely used cognitive architectures (besides Soar). In addition to the provided functionalities, this might result from the fact that the software is free of charge and pretty well documented. Furthermore, a lot of applications exist, where ACT-R is used to model (human-like) agents interacting with humans in virtual or real environments.

The cognitive architecture ACT-R is implemented as a **production system** and provides two different kinds of knowledge representations, declarative and procedural knowledge. The declarative knowledge is represented by chunks consisting of a unique name and attributes (so-called slots). Each attribute has a value that can be a chunk again. Here, the value of the special slot **isa** (obligatory for every chunk) defines a chunk’s type. Chunks are used and changed by the procedural knowledge which is represented by production rules consisting of a condition and an action part. The architecture consists of several modules, which are related to certain parts of the human brain. The long-term memory of the architecture contains the productions and is con-

nected to different buffers storing in every case at most one chunk. Each of the buffers, which build the working memory of the system as a whole, is connected to a certain module with different characteristics. Some of these modules are the visual module for visual input, the manual module executing actions, the intentional module creating imaginations, and the declarative module storing facts. If a model (consisting at least of some chunks and productions) is executed, a main cycle of matching, selection, and execution is performed. Accordingly, at first it is checked which productions have a condition part matching to the current state of the buffers. Then, a conflict resolution is used to select one production which is executed afterwards. This can change the buffers (e.g., changing the contained chunks or requesting new chunks from the corresponding modules etc.) and a new cycle is started.

The architecture provides also several mechanisms in order to **learn chunks and productions**. Basically, chunks are stored in the declarative memory if they are deleted from the buffers (after the action of production was executed). If this occurs several times for the same chunk, the chunk's activation can be enhanced, as a subsymbolic component of the architecture. Furthermore, also utilities of productions can be changed by rewards resulting if a desired or non-desired state of the model is reached (reinforcement learning). Finally, new productions can be created by combining the condition and action parts of two productions if the action of the first production leads directly to the conditions of a second production (production compilation).

ACT-R is utilized in many applications for different purposes. In [FKHB06], it is part of an interaction infrastructure. The developed system provides a software framework for teams of robots and humans. Here, ACT-R is used as a spacial reasoning agent. The teams can contain several robots as NASA's humanoid robot Robonaut (see Fig. 2.10(a)) and several human astronauts. Another application, where ACT-R is also implemented to Robonaut, is presented in [SPS⁺04]. Furthermore, other robot platforms (see [BL06]) are used for implementation as the Pioneer series of MobileRobots, Inc. Here, also the modeling of synthetic agents interacting with real human operators in a virtual environ-



(a) NASA's Robonaut [FKHB06]



(b) UAV STE [GBG⁺05]

Figure 2.10.: Technical examples using ACT-R

ment is described. This kind of application is also typical in the aviation context. For example, ACT-R is used in [BMH⁺09] to model a synthetic team mate acting as the pilot of a UAV within a Synthetic Task Environment (STE) (see Fig. 2.10(b)).

Besides the selected examples, ACT-R is also used in several other systems. However, it can be observed that it is often used as a component of other software. Here, it is used to model, reason, and learn a certain part of the system's environment.

2.4.3. Ripley the robot

The demonstrator **Ripley** is a robot arm with 7 degrees of freedom, force feedback actuators, and a gripper (see Fig. 2.11). The attached sensors are force-sensitive touch-sensors in the gripper, joint angle encoders, and a camera. The system interacts with a table with colored objects and a human for conversational interaction. Based on commands or questions of the human, the robot answers with spoken language or performs actions as 'turning to a certain direction' or 'gripping a certain object'.

The control of the robot is based on a modular architecture containing a so-called **Grounded Situation Model (GSM)**. The GSM is used to represent the current situation of the system. It is structured hierarchically and it is updated by the architecture. On the top level the situation contains 'agents' and 'agent relations'. Each agent is further detailed by 'physical objects' and 'object relations'. Besides the physical objects, agents can also be described by 'mental constructs' connected to the physical world by an interface. The objects consist of several 'properties', e.g., a ball on the table is represented by the properties **position**, **shape**, etc. The properties again are described by 3 different layers for 1) stochastic, 2) continuous, and 3) categorical representation. As



Figure 2.11.: Ripley the robot [Mav07]

a fourth layer, the interaction with the sensing and acting modules of the surrounding architecture is considered. Further information about the GSM and the modular architecture can be found in [Mav07] and [MR06] respectively.

Learning is realized by the storage of so-called events. An event occurs if a certain parameter, e.g., the speed of an object, changes. In order to represent these changes, the current and previous states of the GSM are stored as moments and combined to an event. By storing lists of events, the system is able to remember previous states and answer questions related to the past.

Experiments show that the system interacts with a human through natural language, which is considered as the main problem to be solved. As demonstrating example scenario, the ‘token test for children’ [Di’78] is used. The test contains several steps and is originally used to help children with conversational problems. Here, the successful handling of the first two steps is shown. Furthermore, experiments show qualitatively that the system performs well. Although it is described that the GSM can also be implemented to other kinds of systems, this is currently not demonstrated. Furthermore, other learning mechanisms besides the storage of events are not presented.

2.4.4. SOM-based cognitive architecture

The first concept of an architecture for cognitive autonomous systems based on Situation-Operator-Modeling [Söf01c] as underlying approach was originally proposed by SÖFFKER [Söf01a, Söf01b]. Subsequently, in [Ahl07] (see also [AS08]), the implementation to a real technical system was presented for the first time. The realized **modular architecture** (see Fig. 2.12) consists of a module for basic control, which may be an arbitrary technical system, and a module for cognitive-based control. The sensor measurements of the technical system to be controlled are filtered by a prefilter module which has to be configured by the system designer. The resulting ‘prefiltered situation’ containing a set of ‘characteristics’ describing the current state of the real world is further interpreted by an interpretation module. The resulting ‘interpreted situation’ is a subset of the prefiltered situation with respect to the situation’s characteristics. The cognitive control is realized by the cognitive functions of learning, testing, exploration, planning, and plan supervision, which are realized as algorithms using and modifying a knowledge base. As an example application, the cognitive architecture learns from a mobile robot’s interaction with the real world. In the scenario, the architecture learns how to generate successful plans for the execution of pick and place tasks in a laboratory environment.

The knowledge of the system is represented by C++ objects stored in an **object-oriented database**. The central data structure termed ‘experience’ is used to store the initial situation, the name of the executed ‘operator’ (models an action of the system), and the resulting final situation. Furthermore, different ‘interpretations’ for the initial

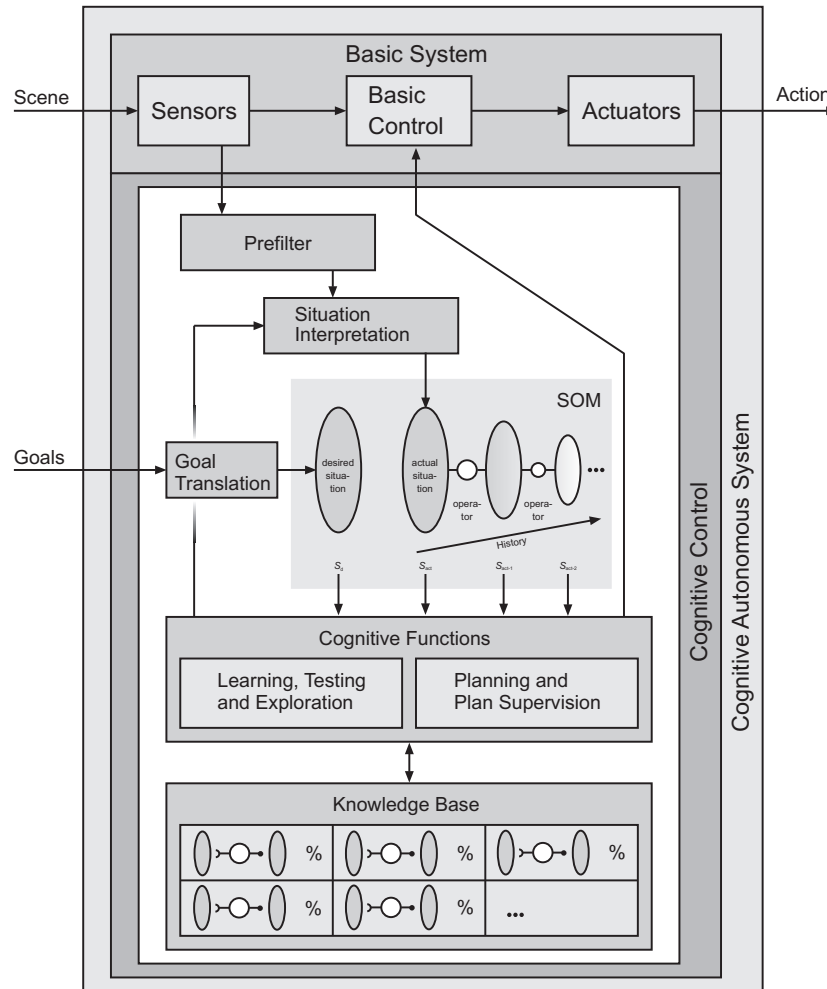


Figure 2.12.: Architecture of the cognitive autonomous system [Söf01a, Ahl07]

and final situations are also stored. In order to acquire new knowledge, **four different levels of learning** are proposed, which include the

1. changing of the ‘transition probability’ of experiences,
2. application of heuristics to generate new interpretations,
3. changing of parameters within the prefilter, and
4. creation of new interpretations by the combination of other interpretations through genetic algorithms

(see Fig. 2.13). From the proposed four levels, the first two levels are implemented and tested. By changing the transition probability, it is stored how often a certain experience was performed successful in the real world. Hence, the system can detect successively which experiences and contained operators respectively have the highest chance of success if the system has the choice of several alternative operators. However, here, unusual

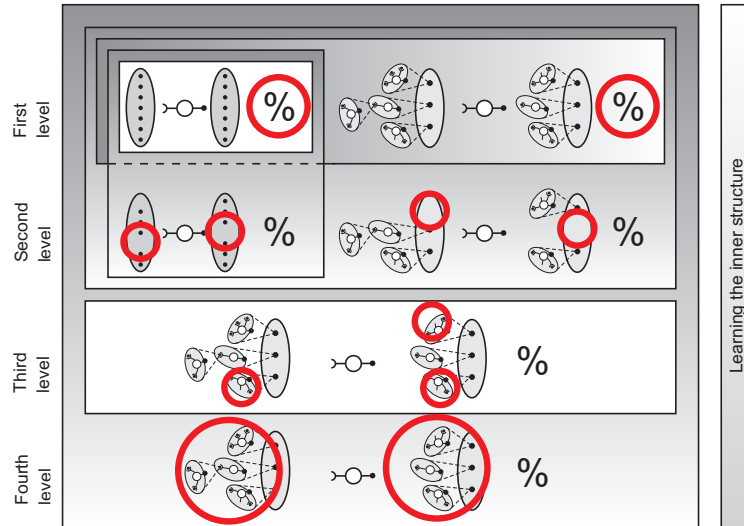


Figure 2.13.: Different levels of learning [Ahl07]

experiences with a low transition probability are ignored although they might be useful in certain (infrequent) situations.

The generation of *plans* (sequence of operators from the current to a given desired situation) is realized by graph search algorithms as breadth-first-search or depth-first-search. If no plan can be found due to a lack of suitable experiences in the knowledge base, heuristics (see [HR83, Sal85]) are applied to generate new experiences. Through this process, the interpreted initial and final situations of the known experiences are modified by adding and/or deleting of characteristics.

The proposed architecture implements Situation-Operator-Modeling as meta-modeling technique and representational level for the first time and offers a promising basis for the realization of real autonomous systems. Furthermore, the cognitive control of a mobile robot in a laboratory environment was presented. Nevertheless, there are several **open issues** which have to be considered for technical systems interacting with real world environments:

- In the proposed approach, an infrequent experience gets a low transition probability if its initial situation is changed by other observed experiences with the same operator more often in another way. Nevertheless, this experience might be useful in some situations. Hence, since an experience with a low transition probability is automatically ignored, the system loses action alternatives and flexibility. Furthermore, the ‘blind’ application of heuristics to complex situations may lead to a relative large number of iterations in order to identify a correct hypothesis, which moreover does not have to be the optimal one.
- The planning function considers recent experiences in the same way as experiences which were stored a long time ago. However, for the interaction with dynamical

environments, this differentiation should be taken into account more stronger. Furthermore, since the used planning algorithms are only applicable to determine the shortest path regarding the number of operators, it is not possible to estimate paths regarding other criteria (e.g., time, energy, safety, etc.), which are essentially more important for technical applications.

- The usage of only one kind of memory (as the proposed object-oriented database) might fail for complex systems since with increasing number of stored experiences, calculation routines for planning and learning takes too much time. Due to the fact that not all information is necessary in each situation, a further structuring (e.g., with different levels of abstraction) would be helpful in order to reduce the complexity.
- Due to the fact that the characteristics of the prestructured situation and the relations of the interpreted situation are set by the system designer, the system lacks autonomy. Hence, the automatic adaptation of parameters and recombination of relations as proposed by the third and fourth levels of learning would be helpful. In addition to that, also the generation of completely new relations and the automatic selection of sensor measurements should be possible.

The results of the described work provide a good basis for further investigation. However, the mentioned improvements can not be realized by simple modifications or new modules since they have influence to the whole architecture and therefore require also a conceptual revision.

2.5. Summary and conclusion

In the previous sections, an overview about the field of Cognitive Technical Systems is given. The sections include definitions of central paradigms and detailed descriptions of important approaches regarding knowledge representation, reasoning, and learning. Furthermore, some examples for integrated cognitive architectures combining several models and mechanisms are described since it is commonly assumed that only such systems are able to model and simulate the variety of human cognition.

In this regard, several different approaches with own concepts of the system's structure exist. Classically, cognitive architectures (especially those that are related to Cognitive Psychology) propose a few kinds of predominantly explicit representations. However, the trend goes to systems representing knowledge explicit and implicit by several kinds of representations (e.g., production rules, neural networks etc.). Nevertheless, some basic paradigms as the step ladder model or the differentiation among working memory, long-term memory, and short-term memory can be found in a large number of approaches. Another trend is that the source code of components which are typically part of cognitive architectures (models, learning methods etc.) as well as the source code of whole

architectures itself (CLARION etc.) are available for free. Hence, they can be modified, tested, and easily integrated into other applications.

Classical cognitive architectures are typically intended to model human performance. However, they are more and more applied to control technical systems (sometimes as a part of another system). Furthermore, some of the younger cognitive architectures are basically intended for this kind of application. Typical demonstration examples are mobile robots, humanoid robots, or software agents. In simulations or real experiments, architectures' functionalities as the solution of certain problems, the acquisition of knowledge, or the communication with humans are shown. However, the format of used memory is often aligned to a certain application and the system borders are typically well defined.

It can be concluded that all presented approaches might be useful to solve some important problems and reach some research goals. This could be the communication by spoken language with humans, the exact modeling of human behavior, or the learning of how to interact with a certain environment. However, in order to develop a general purpose system, which can enable different technical systems to interact autonomously with their environment, research should be more related to

- fundamental methodical approaches for the unified formalization of real world's structure and dynamics,
- application-independent implementation to arbitrary kinds of technical systems, and
- realization of real autonomy by flexible representations of knowledge and learning mechanisms.

Although some of today's systems solve some aspects such as embodiment and learning very well, a lot of approaches neither consider them nor realize them in a flexible manner. In summary, no system or concept exists which can fulfill all the mentioned issues sufficiently.

Although architectures or parts of it are partially recycled, each approach proposes different models, functions, and system structures. However, the development of new candidates of such systems require a lot of work and time. In contrast to that, approaches with variable structures and the ability to add and modify new models and functions more easily are a promising alternative. Consequently, to handle the arising complexity and to simplify the communication between system designers, specialized **meta-modeling approaches** are necessary. Such approaches should be able to formalize the interaction between cognitive systems and real world environments as well as the representation of that by corresponding mental structures in a unified manner.

The development of many architectures is neither intended to **control technical systems** nor driven by experiments in real world environments. In this respect, the

question arises whether a model of human performance can be transferred one by one to a technical system whose interaction is based on totally different requirements regarding hardware and information processing in general. Furthermore, in the case of architectures that are developed in order to control a certain technical system, the used representations and functions are often specified to the related hardware and application respectively. Although this can lead to successful results, the learning of unexpected facts and relations is avoided.

Although many systems provide **learning mechanisms**, it seems to be sometimes not intended from the very beginning. However, learning is a fundamental characteristic of cognitive systems, influences their structure, and can not be simply added. Hence, this might be the reason why some systems only provide simple learning mechanisms as the storage of events or experiences respectively. However, for the realization of real autonomy, it is moreover necessary that all cognitive functions and used models are influenced by learning. Especially, learning regarding a system's recognition and attention capabilities is often missing.

This thesis describes an approach considering (in contrast to other approaches) all mentioned demands for representation, learning, and embodiment, in order to realize Cognitive Technical Systems. Here, Situation-Operator-Modeling is applied as methodical background to realize a framework consisting of several models and mechanisms, which can be used to realize an integrated cognitive architecture based on a unified formalization of the real world. The approach is related to arbitrary technical systems enabling the portability of knowledge and functionalities. The development is driven by experiments with a mobile robot, whereas human cognition serves as inspiration for the realization of a flexible and adaptive system behavior. The internal representation is hierarchized in two dimensions and realizes the situational and task-relevant reduction of complexity. Furthermore, knowledge can be illustrated clearly and is therefore accessible by humans. All cognitive functions use models that can be extended and refined from interaction.

3. Modeling of human interaction

In this thesis, the Situation-Operator-Modeling approach [Söf01c] is applied to formalize the interaction of agents (humans, robots, etc.) with other agents or objects. In order to develop an executable model which can be used for simulation and analysis, high-level Petri Nets (HPNs) are used for implementation.

The SOM-based analysis of Human-Machine-Interaction is visualized in Fig. 3.1. As the first step, the interaction between a human operator and a technical system is qualitatively described by the SOM approach. Here, the perceived scenes are modeled as situations and the performed actions are modeled as operators. Furthermore, certain behavior patterns as typical human errors can be defined [Söf01c, Söf04b]. In order to simulate and to analyze this interaction, the resulting SOM-based model can be described by patterns of high-level Petri Nets which can be implemented by special software tools. Besides graphical modeling and different kinds of simulation functions, some of these tools also provide the automatic generation of a state space containing all states and

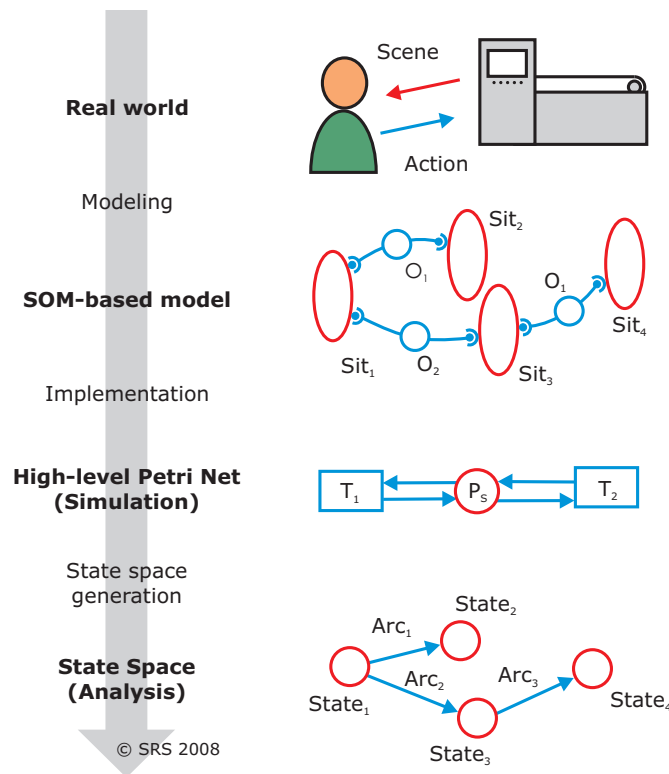


Figure 3.1.: From modeling to analysis of Human-Machine-Interaction

state transitions of the Petri Net. Since the complete or relevant parts of a Petri Net's dynamic can be represented by the state space, it can be used to analyze the considered Human-Machine-System. Therefore, search algorithms or query functions can be used, e.g., to detect defined human errors (see [GOS09, GOS08]), etc.

Due to the fact that Situation-Operator-Modeling, the implementation with high-level Petri Nets, and the state-space-based analysis build the technical core of the proposed new approach, the relevant parts of each step are detailed in the following sections. In Section 3.1, methodical basics of Situation-Operator-Modeling, the SOM-based formalization of a typical human error behavior, some details about the hierarchy in SOM-based models, and a technique for the computer-aided visualization of SOM-based models are described. After that, Section 3.2 illustrates the implementation and simulation with high-level Petri Nets. Therefore, the similarities between SOM and HPN as well as the two example patterns for SOM-based Petri Nets are presented. In Section 3.3, state spaces of HPN models, their automatic generation, and their analysis are described. Finally, the automated detection of the human error 'rigidity' is illustrated in order to give an example for the presented approach.

3.1. Situation-Operator-Modeling (SOM)

Situation-Operator-Modeling (see [Söf01c, Söf08]) can be applied in order to model Human-Machine-Interaction. In the following paragraph, the structure of a SOM-based model and the functions of its elements are described briefly. Afterwards, Section 3.1.1 describes the formalization of human errors as examples for certain behavior patterns. Finally, the hierarchy within a SOM-based model as it is used in this thesis is detailed in Section 3.1.2 and Section 3.1.3.

Within the SOM approach the processes in the real world are considered as sequences of scenes and actions, which are modeled as situations (time-fixed description of the considered system or problem) and operators (changes within the considered system) respectively (see Fig. 3.2). A situation s_i consists of relations r_i and characteristics c_i with the parameters p_i (values of the characteristics). In technical systems, the characteristics can be physical quantities measured by the sensors. The relations represent an internal structure of the situation by linking the characteristics to each other through arbitrary functions. The operators o_i have the same quality as the relations of the situation. An operator transfers a situation to another ($o_i : s_x \rightarrow s_y$). Depending on the operator's function F , the characteristics, the characteristic's parameters, or the relations can be changed. The condition, whether an operator can be applied, is described by the operator's assumptions. An operator on a higher hierarchical level can be built by the combination of several operators termed as meta operator ($o_{i \rightarrow n} : s_i \rightarrow s_n$).

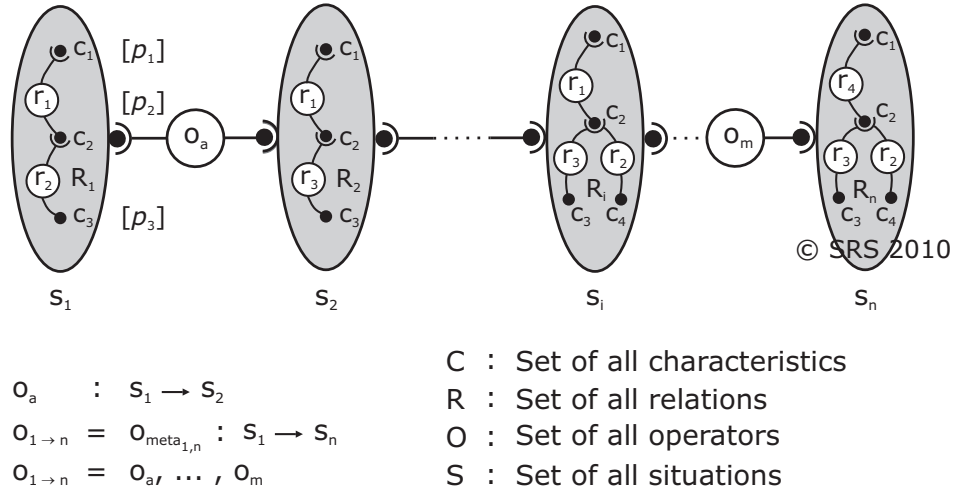


Figure 3.2.: SOM-related notation and graphical representation

Although Situation-Operator-Modeling is related to logic-based approaches as the classical situation calculus [McC63], which assumes also discrete changes of the real world, it is more than a certain kind of knowledge representation. Besides the pure representation of knowledge, SOM can also be used as meta modeling approach in order to structure the complexity of human interaction and mental representations of cognitive systems as well.

The SOM approach includes also a graphical representation, in which situations are illustrated by gray ellipses with black dots denoting characteristics and white circles denoting relations (see Fig. 3.2). Such as the relations (or passive operators), also the active operators are represented by white circles. As shown in the figure, certain parts of the complete model as the outer connections of situations and operators or the situations' internal structure can be also represented as graphs if they are considered separately.

3.1.1. Formalization of human errors

The SOM approach can also be used to formalize human errors [Söf01c, Söf04b]. According to DÖRNER [Dör97], human errors are classified with respect to the interactions of humans in complex dynamical systems. Coming from psychology, word models are typically used to describe and to distinguish different human errors, which are divided into four main clusters: goal elaboration, decision processing, control of the actions, and errors due to internal cognitive organization problems. The term 'human error' is used in psychology (e.g., [Dör97]). In general, the term 'error' describes differences to a defined desired behavior etc. In this regard, 'human error' is also used to describe behaviors which are not optimal (regarding a certain quantity) and which lead to undesirable situations respectively. Nevertheless, these kinds of behaviors could also be useful strategies in certain situations.

As an example, the human error ‘rigidity’ is described and visualized with SOM notation in Fig. 3.3 (see also [Söf04b]). The situations (s_i) including the characteristics (a_i , b_i) and relations (r_i) are represented by gray ellipses and the operators (o_i) are represented by white circles. In the situation trajectory depicted in the upper part of the figure, the desired situation s_2 is not reached as planned in the upper part, due to external effects and disturbances. Instead, a different and unexpected situation s_{2a} is resulting. Hence, o_2 will not lead to the desired goal due to the changed situation s_{2a} . The human error rigidity includes that the known and previously planned operator o_2 is nevertheless realized inconsiderately by the human operator, although the assumptions for its application are no longer fulfilled.

Besides the classification according to DÖRNER, human errors are also classified and informally described in [Rea90], [Hac05], and [RR83]. In each classification, different aspects of human behavior are focused, but some of the error descriptions from different classifications are nevertheless very similar to each other (although termed differently in some cases). For example, in the classification according to REASON, also ‘slips’ and ‘lapses’ are considered besides rule-based and knowledge-based mistakes. Slips and lapses are errors related to the skill-based level of human behavior and not considered by the classification of DÖRNER.

Independently whether the description of a certain behavior is considered as erroneous or desired, these word models may be formalized by Situation-Operator-Modeling and detected automatically based on a state space analysis (as shown by the example above). However, this procedure has to be investigated for each single description individually.

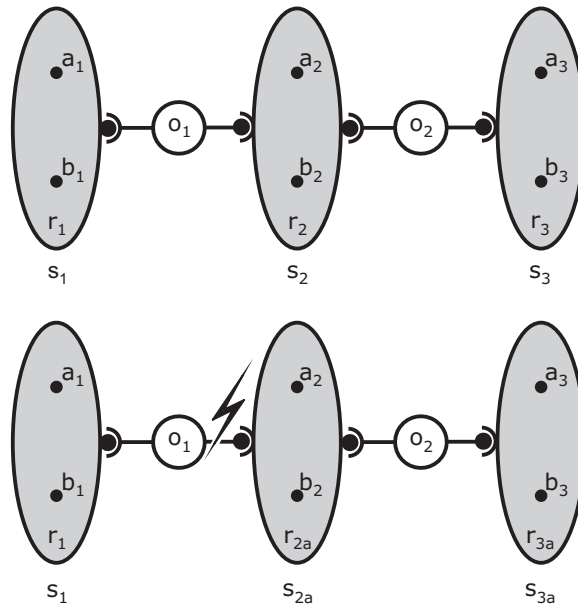


Figure 3.3.: SOM-based formalization of the human error rigidity [Söf04b]

In [Has08], e.g., the formalization and state-space-based detection of the human errors methodism, thematic vagabonding, slips, and lapses are presented.

3.1.2. Hierarchical relations among situations

SOM situations are characterized by an internal structure and they can be considered as hybrid and open vectors of characteristics. This means that the characteristics can have different datatypes and that the number of characteristics related to a situation does not have to be fixed, which is a key feature regarding learning [Söf01c]. Furthermore, the types and number of relations can also vary. The situation plays a central role for the information processing in this thesis. Hence, it is necessary to define when two situations are equal or if not, in which way they are related to each other. Through the hybrid and open character of situations, the equality of situations and their hierarchical relations can be defined in different ways. However, in the context of this thesis, it is sufficient to describe those aspects from the characteristics point of view. Thus, the equality of situations and two different relations among situations are defined as follows.

A situation is related to a certain set of characteristics. Each characteristic is defined by a unique name (or identifier) and a variable parameter (or value) that is related to a certain datatype. Furthermore, one characteristic can be contained in different situations, however, not several times in one situation. Accordingly, it is defined that two situations are equal if they consist of the same set of characteristics and if all same characteristics have the same parameters.

If two situations have different characteristics or different amounts of characteristic, they can not be compared in the described manner. Nevertheless, two situations that do not fulfill the defined condition of equality, can represent one and the same scene anyway. Due to the fact that these relations play an important role in the next chapter of this thesis, two different relations of situations describing one and the same scene are defined and visualized in Fig. 3.4. In both cases, the left situation is more special and the right situation is more general. Hence, the following definitions describe hierarchical relations.

The first hierarchical relation is illustrated in Fig. 3.4(a). The set of characteristics of the right situation is a subset of the set of characteristics of the left situation. Hence, the corresponding scene is described by the right situation more generally than by the left situation. If such a hierarchical relation exists, the more general situation is denoted as ‘subsituation’ of the more special one (see Def. 3.1).

Definition 3.1: Subsituation

If a situation s_a contains the same but less characteristics than a situation s_b , the situation s_a is termed a ‘subsituation’ s_b^{sub} of the situation s_b .

The second hierarchical relation is shown in Fig. 3.4(b). Here, the left situation is illustrated with its internal structure in a combined view. The parameter of the

third characteristic depends on the parameter of the first and second characteristic and describes therefore the same information in another or more abstract manner. If the characteristics are divided into different situations (as shown in the figure), also a special and a more general view to the same scene arises. According to Fig. 3.4(b), the middle and right situations are subsituations from the left situation and the right situation is a ‘derived situation’ from the middle situation (see Def. 3.2).

Definition 3.2: Derived situation

If the parameters of a situation s_a ’s characteristics depend on the parameters of situation s_b ’s characteristics, the situation s_a is termed ‘derived situation’ s'_b of the situation s_b .

Due to the given definitions, situations can also be compared if they are linked in a hierarchical manner. If a situation has to be compared to a subsituation, it has to be simply checked if the parameters of the subsituation’s characteristics are equal to the parameters of the corresponding characteristics in the situation. If a situation has to be compared to a derived situation, the corresponding relations have to be applied to the situation in order to estimate whether the resulting characteristics have the same parameters as the characteristics of the derived situation.

3.1.3. Action spaces

According to the SOM approach, the possible interaction of an agent (human, robot, etc.) can be illustrated by a net of scenes and actions or from a modeling point of view by a net of situations and operators. This kind of view does not consider the general

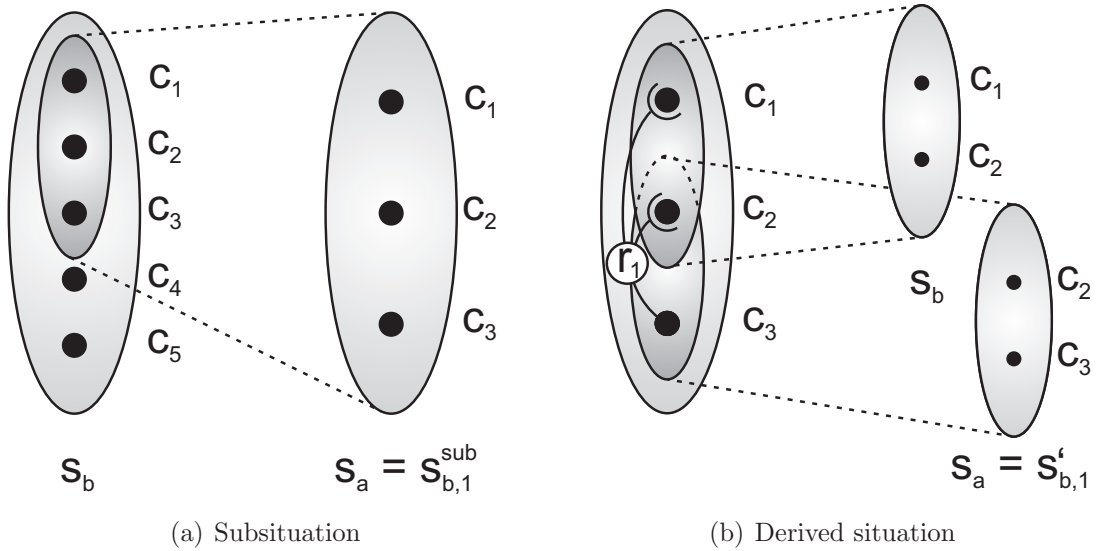


Figure 3.4.: Hierarchical relations between situations

functions and assumptions of operators. The net is related to a certain initial situation and can contain the same operator several times. In order to generate such a net, all possible operators have to be applied to an initial situation. Then, new situations result, which have to be changed again by the operators and so forth. Due to the fact that the resulting net represents the interaction within the real world, the net is denoted in the following as ‘action space’ (see Def. 3.3).

Definition 3.3: Action space

The term ‘action space’ denotes a set of situation-action-sequences resulting from the propagation of alternative actions from a certain initial situation.

If a general model of an agent’s possible operators exists, the general model and the current situation could be taken to generate an action space. However, this net becomes very large and complex if an agent can perform a lot of actions and/or if moreover the action’s of other agents are taken into account. Hence, the complexity of the considered action space has to be reduced in a suitable manner. Besides the current situation which is the origin of the action space, also the current task of the agent defined by one or several goals can be used to neglect irrelevant aspects.

If an action space is generalized, groups of the contained situations are represented by subsituations (or derived situations). Hence, the subsituations also represent a certain subset of the previous action space. This subset represents the interaction that is possible as long as the corresponding subsituation is valid and ends if the subsituation changes. The connections between the subsituations are meta operators, which are sequences of the previous action space’s operators. They can be directly derived from the interaction, which is represented by the meta operator’s initial subsituation. The resulting general action space is termed as ‘meta action space’ (see Def. 3.4). Of course, the meta action space can be generalized, too. Then, the resulting action space is termed as meta action space of 2nd order, which is related to operators which are defined as the system’s basic operators. All meta operators can be considered as virtual abstractions of the basic operators.

Definition 3.4: Meta action space

The term ‘meta action space’ denotes a general version of an action space resulting from the propagation of alternative actions from a certain initial situation’s subsituation.

In order to take the agent’s tasks into account, the defined subsituations of the meta action space can be related to the agent’s goals if these goals are defined by subsituations (see Fig. 3.5). However, the behavior of a complex system can be hierarchized in different ways. For example, also other characteristics which are not contained in the goals can be added to the subsituations to abstract regularly occurring action pattern.

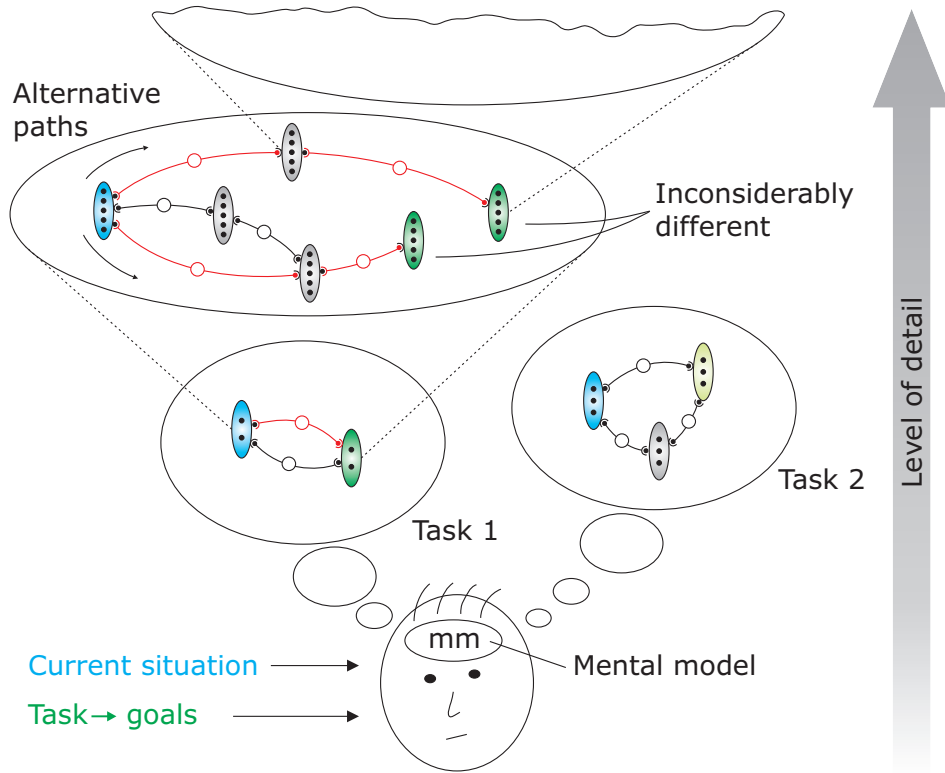


Figure 3.5.: Hierarchization of action spaces

As an example, the usual working day of an arbitrary employee is considered. The day starts in the morning, when she/he is at home. Then, the employee drives to her/his office (or workshop etc.) in order to do her/his work. Finally, she/he drives back home. This action space contains two operators ($O_{\text{driveToWork}}$ and $O_{\text{driveHome}}$) and two situations with the characteristic c_{atHome} . In one situation, the parameter of the characteristic c_{atHome} is **true** and in the other situation the parameter is **false**. An action space with a lower degree of abstraction can describe how to come from home to work. This action space is more complex with several alternatives, depends of course on the current situation, and ends if the characteristic's parameter of the subsituation (here: c_{atHome}) changes. Hence, the employee (who is able to perform a broad variety of different interaction patterns) only considers the situation- and task-relevant actions and characteristics. If the employee is in vacation, the action space at the top level and all lower ones can have a totally different structure and a lot of knowledge (not everything) that is necessary for her/his work does not have to be considered.

3.2. Simulation of SOM

The introduced SOM technique provides a qualitative modeling approach starting from a symbolic notation to structure and investigate human interaction with technical systems.

In order to analyze complex systems with many degrees of freedom, computer-based representations of SOM-based models have to be built and simulated. The computer-aided simulation of SOM-based models has formerly been realized in the form of both textual programming languages [AS08] and high-level Petri Net (HPN) formalisms [Gam06].

3.2.1. Implementation with high-level Petri Nets

High-level Petri Nets combine the advantages of a graphical representation with the benefits of a formal executable model. According to [Bau96], HPNs are defined as a class of net formalisms, which extends the classical Petri Net formalism originally introduced by PETRI [Pet62] (consisting of places, transitions, arcs, and black, uniform tokens) mainly in that HPNs allow several different tokens (such as numerical values, data structures, or complete software objects) at the same time on the same place and often support net hierarchization.

Between SOM and HPNs certain structural similarities exist. They are both inherently bipartite formalisms consisting of an active element representing the system functionality and a passive element representing the system state. In this respect, a natural and intuitive correspondence exists between the notion of operators in SOM and transitions in HPNs. The meanings of the term situation in SOM and the term place (in combination with one or several tokens) in HPNs are also closely related. Additionally, some Petri Net software are provided with a well developed repertoire of basic graph theoretic analysis techniques (e.g., [JCK06]). These techniques are useful to prove individual properties on SOM-based models and they allow analyzing interaction described by SOM-based approaches.

Nevertheless, the SOM approach goes further than the presented method using Petri Net patterns. This becomes apparent if the modeling of open systems is considered. In such cases, the learning of the model or parts of it should be possible. However, for complex systems (as Human-Machine-Systems) the simple adaption of parameters (or add some markings on a place) is usually not enough. In fact, the model sometimes has to be restructured, to map the new dynamics of the considered system adequately. A further discussion of this issue can be found in the next chapter.

3.2.2. Example for a SOM-based Petri Net

Two different basic net patterns are briefly presented in this section, which define a correspondence between the elements situation, operator, characteristic, relation, and assumption of a SOM-based model and the terms place, transition, arc, and token of a HPN model. In these patterns, relations r_i and operators o_i are assumed to be time invariant. Thus, similar patterns can be realized with most discrete HPN formalisms, including Coloured Petri Nets [Jen97, JKW07] and Reference Nets [Kum02]. Both formalisms were formerly used to model systems as the ones focused in this thesis. Espe-

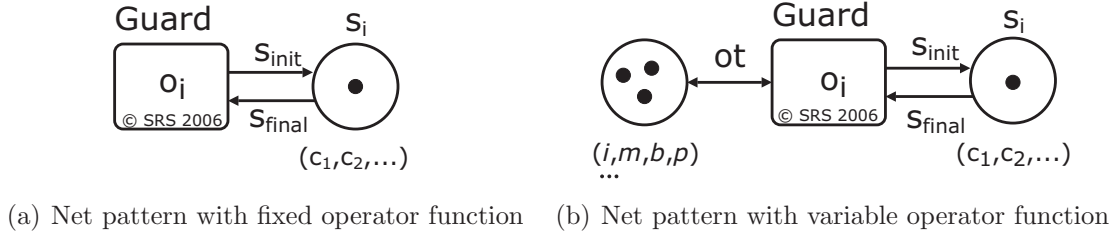


Figure 3.6.: SOM-based net patterns

cially, Coloured Petri Nets are frequently applied to model Human-Machine-Interaction and Cognitive Systems (e.g., see [Wer06, MOW08]). In contrast to that, Reference Nets are more often applied to model Multi-Agent-Systems (e.g., see [vLMV⁺03]).

In the first pattern, the situation s_i is represented by a place containing a single token. The token is a data structure potentially consisting of different data types, representing the characteristics C_i of the situation s_i . The function of the operator is represented by fixed code of the transition (fixed operator function) with the initial situation s_{init} as input and the final situation s_{final} as output. The assumptions IA_i and EA_i for the execution of a certain operator o_i are validated by the guard function and the actual bindings of the transition. The resulting HPN structure is shown in Fig. 3.6(a).

In the second pattern, the situation s_i is also represented by a place, containing a single token with the characteristics of the situation s_i . However, in contrast to the first pattern, here, a transition represents a group of operators with similar structure and function, while the exact quantitative effects of the individual operator are specified in an operator token ot (variable operator function). The operator token has a format (i, m, b, p) consisting of constraints (specifying assumptions EA_i and IA_i when the operator can be used) and instructions (parameters for modifying the characteristics). Through handling of the basic operators as dynamic tokens, learning of operator effects can be realized within certain narrow bounds as long as the basic structure of the operator o_i is known beforehand. The resulting structure for this net pattern is shown in Fig. 3.6(b).

Besides the described kind of patterns with time invariant relations and operators, also patterns with continuous parts are possible if this is provided by the used simulator. For example, the software Renew [KWD08] allows the parallel execution of the Petri Net and Java threads simulating continuous processes. More details about these kind of patterns and the simulation of SOM in general can be found in [Gam06].

3.3. State space analysis

State spaces of HPN models can be represented as discrete, directed graphs (digraphs). The nodes of the graph represent different states (markings), which the model may obtain, starting from a certain initial state. The arcs of the graph represent possible transitions between these states. Adopting an appropriate implementation technique and net patterns to model SOM in HPN software as presented in [Gam06], the states are interpreted as SOM situations, while the state transitions are associated with SOM operators. Depending on the properties of the model and available computational resources, the resulting state spaces may be complete (representing all reachable states) or only partial (representing a subset of all reachable states, e.g., within a certain search depth with regard to the initial condition). Accordingly, it becomes possible to automatically determine a (partial or complete) set of reachable SOM situations and calculate possible SOM-operator-sequences between these situations.

State spaces can be generated automatically in some software tools providing moreover a large repertory of query functions which can be used, e.g., to determine states with certain properties or a shortest sequence of firing transitions between two states. However, if the automatic generation of a state space is not provided by the software tool to be used, the state space can also be generated by the model itself. Therefore, the effects of all modeled operators related to an initial situation have to be calculated. Then, the same calculation has to be done with the new situations resulting from the previous calculation. This procedure can be repeated until a certain number of situations is calculated or until no new situation results from the calculation. The result of a single calculation can be stored in an object with a certain data structure containing the initial situation, the operator, and the final situation. Finally, the state space corresponds to the set of generated objects and it can be analyzed by graph search algorithms.

Applying the calculated space of reachable situations is a helpful approach for the analysis of Human-Machine-Interaction. As an example, the detection of human errors can be investigated since the approach allows to relate the observed actions of the human operator to the (known) set of reachable situations and executable operators of the system. On the basis of the state space, it becomes possible to formally determine desirable (goal) and non-desirable (unsafe) states/situations which can be reached by the HMS and then check if the human operator's actions tend to achieve (come closer to) a certain goal situation. It is also possible to observe if an action sequence serves to pursue one single goal consistently or iterates/jumps between various competing goals etc.

According to Def. 3.3, the calculated space of situations and operators can be denoted as action space. However, it has to be guaranteed that the Petri Net exclusively represents the interaction and does not contain further administrative parts, which could be activated during simulation. Furthermore, if the model is established based on the described patterns in Section 3.2, only one degree of abstraction can be considered. How-

ever, as argued in Section 3.1.3, the complexity of many real world systems can only be managed if the interaction can be described by several situation- and task-related action spaces with different degrees of abstraction. Hence, it is difficult to model the whole interaction of a complex system in one Petri Net, especially if new abstraction levels have to be generated during runtime. A solution of this problem is the usage of several Petri Nets (or action models), which is detailed in the next chapter.

3.4. Example: Automated detection of human errors

This section finally illustrates the application of the whole sequence of modeling, simulation, and analysis to the detection of human errors. Therefore, the interaction between a gamer and an arcade game is considered. As representative example, the human error rigidity (see Section 3.1.1) as a certain behavior pattern can be detected by query functions if the performed actions of the gamer are analyzed with respect to the whole state space of interaction (action space).

The automatic detection of human errors within the interaction of the system can be realized through formal state space query functions. In order to make the functions reusable for different kinds of Human-Machine-Systems, the queries should be built in a manner that the structure of the error detection is generic and remains independent of the specific system. Only the concrete meanings of desirable/non-desirable goal situations in a certain application context are system-specific and have to be exchanged. The presented approach was already realized and tested successfully for different human errors by a simulation environment named HMI Analysis Architecture. In Fig. 3.7, the connections between real world, modeling, and analysis are illustrated.

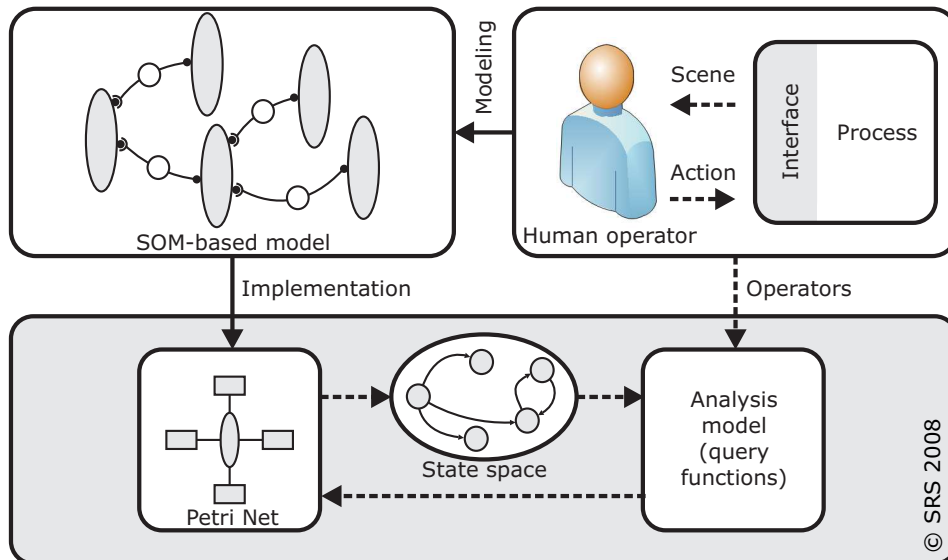


Figure 3.7.: HMI-Analysis-Architecture

Here, the interaction between a human operator and a (real or simulated) process is modeled using the SOM approach. This action model is implemented with HPNs. From the Petri Net describing the interaction of the real world a state space is generated. An analysis model uses the state space as well as the actions of the real world (modeled as operators) as inputs to evaluate the behavior of the human. As an example, the interaction with an arcade game between a human operator (gamer) and a computer (arcade game [Art10]) is chosen. The arcade game enables the design of custom scenarios and it can be replaced by other simulated or real technical processes. The modeling and analysis of the interaction is realized by the software CPN Tools.

As described in Section 3.1.1, the human error rigidity describes a human behavior, in which a human operator adheres rigidly to a previously planned strategy although due to external effects a change would be necessary and more efficient respectively. Through the formalization of rigidity with formal query functions, the automated detection is possible. Therefore, a set of possible final goal situations is calculated using the full state space. From these situations and the observation of user actions, a set of user goal situations (those final goal situations to which the user actions are directed) can be derived. Both, the possible final goal situations and the user goal situations are updated after every observed action. The detection of rigidity itself is based on two conditions. The first condition is fulfilled, if a user action is not directed to a final goal situation and the second condition is fulfilled, if the user action was directed to a previous user goal situation. This implicates the occurrence of a user independent action, which was not detected or ignored by the user corresponding to the definition of rigidity.

In the arcade game (see Fig. 3.8), the human operator (or gamer) has to control the humanly looking agent. It has to pick up at least one emerald and leave the level by reaching the exit door in the lower right corner. Besides the agent controlled by the human operator, there are two hostile monster agents. These agents move autonomously

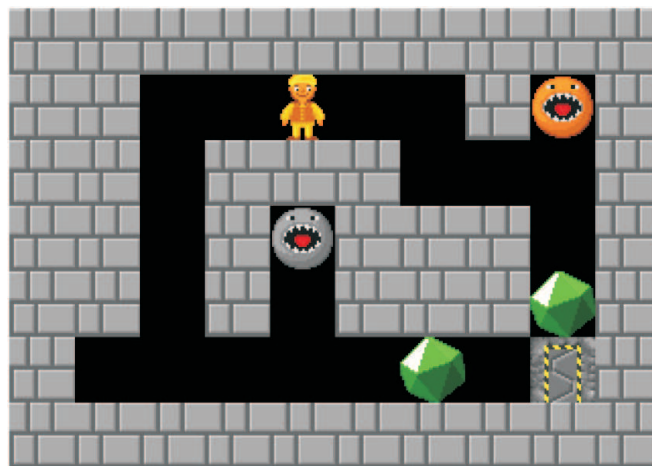


Figure 3.8.: Arcade game (see also [Art10])

and pose a threat to the player's agent exposing different behaviors. The light-colored monster (on the right side) moves to one direction. If it reaches an emerald or a wall it changes the moving direction to the opposite side. The dark-colored monster (in the middle) moves to one direction, too. However, in the lowest position, it decides weather it keeps the vertical moving direction or changes to a horizontal moving direction. Furthermore, the dark-colored monster is able to 'eat' emeralds.

Before the first action, the human operator observes the behavior of both monsters. After these monsters have returned to their initial location (Fig. 3.8), the human operator starts to control the agent to the bottom emerald. The exact sequence of actions observed in this example is then: agent-left (1), agent-left (2), dark-monster-down (3), agent- down (4), dark-monster-down (5), agent-down (6), dark-monster-right (7) and agent-down (8), which is illustrated in Fig. 3.9. After each control action of the human

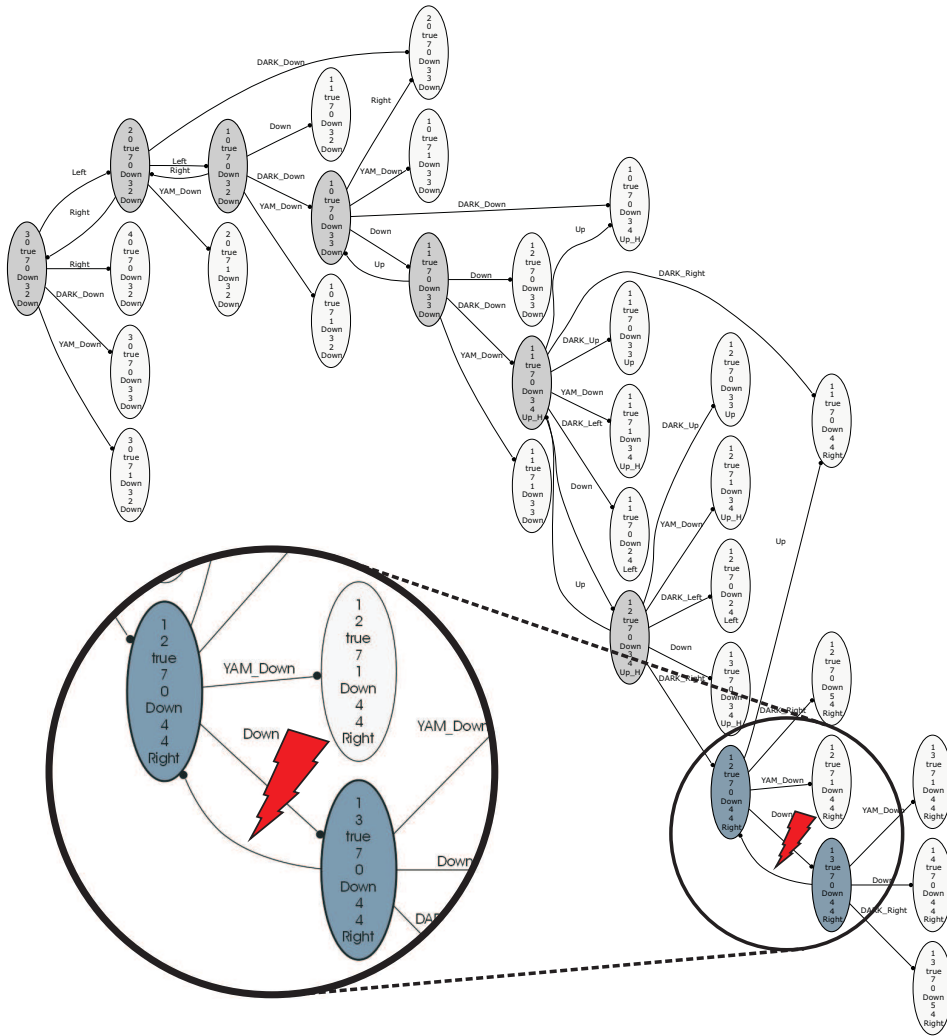


Figure 3.9.: State-space-based detection of the human error rigidity

operator the possible user goals will be determined. Observing the anticlockwise movement of the operator towards the lower emerald the ‘possible user goals’ will initially include all situations where either the user has collected the lower emerald and leaves the level through the door directly, or has collected the lower emerald, then the upper emerald and then exits the level either way. In any case collecting the lower emerald is the obvious intermediate goal state. However, this intermediate goal is no longer reachable after the 7th action (dark-monster-right). Due to the fact that the human operator further pursues the old action plan/goal by performing the 8th action (agent-down) the human error ‘rigidity’ is detected (marked through the lightning in Fig. 3.9).

3.5. Concluding remarks

In the previous sections, an approach for the analysis of Human-Machine-Interaction is presented. This approach is based on a concrete (measured) behavior of a human operator, a formal description of a certain behavior pattern (as proposed in [Söf04b]), and a representation of all action alternatives. The representation of the action alternatives is a state space and the formal description of behaviors is based on word models from psychology (in this case according the classification of DÖRNER). Hence, limited resources or similar cognitive aspects are not explicitly modeled, but derived from the error description and a certain human behavior to be analyzed.

In the field of Human Factors, human behaviors are often investigated based on cognitive models. As an example, WERTHER [Wer06] proposes an integrated model of a Human-Machine-System, which is also based on Coloured Petri Nets. The model can be applied to detect critical situations and estimate reasons for certain erroneous behaviors. However, this approach differs basically to the approach presented in this thesis since the modeling is not based on an intermediate level, as provided by Situation-Operator-Modeling, and it is not intended to detect typical behavior patterns related to human errors.

The proposed SOM-based patterns of high-level Petri Nets are also applied in the following chapter in order to realize an integrated model of human cognition. However, this thesis focuses on the development of a cognitive architecture for the guidance of technical systems. The cognitive architecture is characterized by several learning mechanisms, which usually do not have to be considered if the performance of a well skilled human operator is to be modeled. Nevertheless, the cognitive architecture by itself may also be applied in the future to model and analyze human performance.

4. SOM-based design of Cognitive Technical Systems

The modeling and simulation of cognition and the design of Cognitive Technical Systems is intended by several different approaches (see Section 2.3). These approaches usually propose a fixed system architecture defining the connection and interaction of different modules, models, functions, etc. The connections within frameworks or agent-based systems are more flexible, but they often do not provide a clear structure. In order to allow the design and handling of complex cognitive models, this thesis assumes that system constants have to exist in a general manner and should not limit the modeling capabilities through a specific architecture. Hence, an intermediate level between the real world and the model world is necessary to map the external interaction and internal representation of cognitive agents in a formal and unified manner.

The definition of an intermediate level should be realized by an approach which is not limited regarding a certain kind of implementation and which provides the integration of new concepts and other approaches. Here, the suitable combination of different special representations within a meta model would be helpful since only one kind of representation may not be able to describe the whole variety of human cognition (see also [Min91]). In this regard, it should be possible to structure the complex relations between a cognitive system's action- and perception-based capabilities in an intuitive and unified manner.

In this thesis, an intermediate level is defined by Situation-Operator-Modeling since it provides the described requirements. It can be used for knowledge representation and meta modeling as well. In Fig. 4.1, the basic idea of this thesis is visualized. The figure is divided into three levels. At the top level, the considered aspects of the real world are illustrated. These aspects include the interaction of agents as well as the corresponding mental representation and mechanisms of an involved cognitive system, which are formalized at the intermediate level by the SOM approach. Finally, the resulting formal representation can be transferred into computational models as Petri Nets. The Petri Nets can be simulated (see Section 3.2) or analyzed (see Section 3.3) to realize Cognitive Technical Systems interacting autonomously with their environment.

The following sections describe the development of a SOM-based framework for the design of Cognitive Technical Systems. At first, the implementation of situations and operators are explained in detail. Therefore, high-level Petri Nets and Java classes are used. Then, the development of models based on the defined implementation and the functions processing the models are discussed. Due to the fact that learning is the key

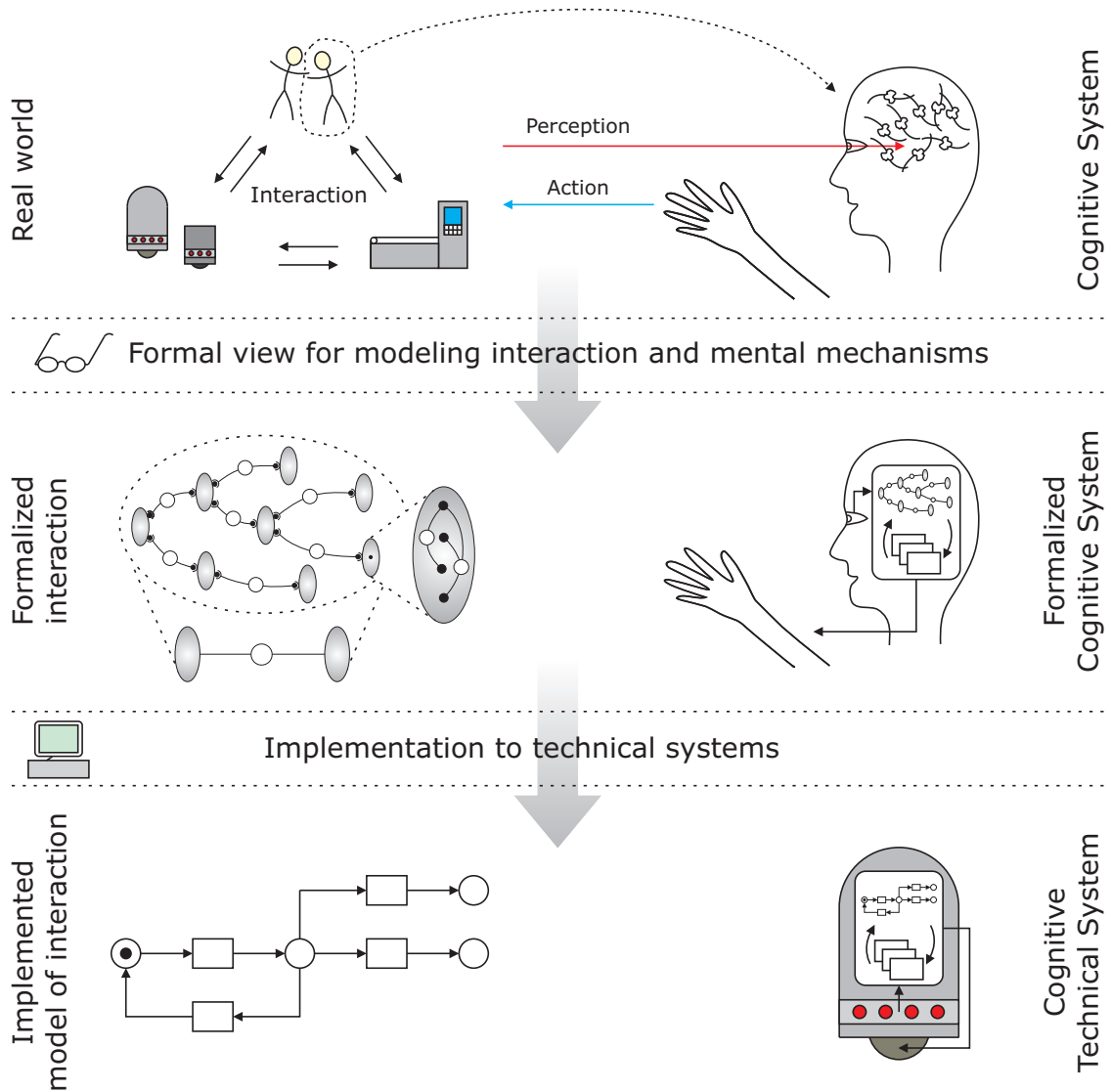


Figure 4.1.: Required intermediate level between real world and technical system

feature of cognitive and autonomous systems respectively (see Section 2), several kinds of learning mechanisms for different purposes are presented. Finally, the ILCA architecture as an example for a cognitive architecture based on the developed framework is illustrated.

4.1. Representational level

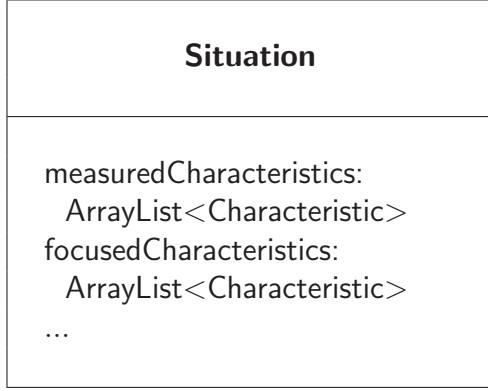
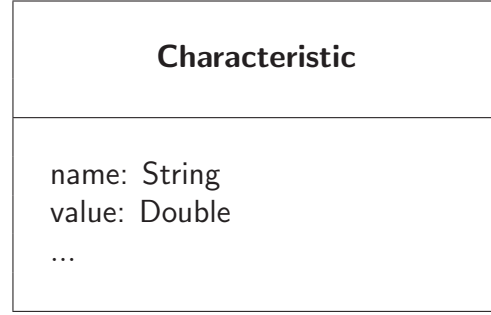
In order to implement SOM, high-level Petri Nets and Java objects are used (see Section 3.2). In the proposed framework, situations and operators are represented by SOM-based Reference Nets implemented with the high-level Petri Net simulator Renew. Reference Nets are based on a net-in-net formalism [Val98] which allows Java objects and whole nets as tokens. Hence, the model is characterized by a hierarchical structure of different nets. These nets can be simulated independently from each other and communicate via so-called ‘synchronous channels’. Furthermore, synchronous channels can be used to control and communicate with other software. One drawback might be the fact that Renew does not provide the generation of a state space and related analysis methods. Thus, these functionalities are realized within the model.

The implementation of SOM-based models and their simulation with Renew works quite well, but comparable Java objects can be processed by algorithms much faster than Petri Nets. Hence, the Petri Net’s functionality can be replaced by a corresponding object-oriented representation if fast calculation is required. Nevertheless, the modeling with Petri Nets is necessary for the development of the framework since it offers an intuitively access for the implementation of SOM.

In the following, the implementation of SOM by Reference Nets and Java objects is explained. The core of the information processing within the proposed framework is the situation. In this thesis, three different kinds of situations are distinguished, which are described in the next section. Then, two different strategies for the representation of the real world’s actions are illustrated. On the one hand, effects which are related to certain situations are represented by so called ‘experiences’ and on the other hand, the situation’s structure as well as the general conditions and effects of actions are represented by nets for passive operators (relations) and active operators. The presented data constructs are used in the following sections below in order to build particular models for certain cognitive functions.

4.1.1. Different types of situations

The key element for the whole information processing of the proposed SOM-based framework is the situation. A situation is represented by the class **Situation** (see Fig. 4.2) contained in the Petri Net model as token. The characteristics of a situation are represented by a list of the class **Characteristic** which consists primarily of the attributes **name** and **value** (see Fig. 4.3). In contrast to [Ahl07], the relations are not attributes of **Situation**, but by the implementation they are linked indirectly through a list of characteristics (assumptions of the relation). The assumptions of a relation define whether a relation belongs to a situation or not (different to previous implementations of SOM).

Figure 4.2.: Class **Situation**Figure 4.3.: Class **Characteristic**

One of the main differences between SOM and other approaches is the ability for open, hybrid, and generic modeling of scenes by situations using internal structuring, which allows to describe a certain time-fixed state of the real world. Hence, it is possible to represent different hierarchical levels of situations and a dynamic change of the considered situation's structures, which correspond to the human capabilities of pattern recognition and attention. Thus, it is possible to reduce the complexity of the represented real world to make the system's input processable and to keep the allocation of resources small. Therefore, the situation considered by the system should be as small as possible and should contain characteristics on a preferably high degree of abstraction.

In order to realize the basis for the capabilities of pattern recognition and attention for technical systems, different groups of characteristics (combined in situations) with respect to their usage are proposed here. In the following,

- measured characteristics,
- derived characteristics, and
- focused characteristics

are distinguished. The connections between the different types of characteristics and situations respectively are visualized in Fig. 4.4.

The **measured characteristics** C_m are closely related to the sensors of a technical system and they have to be defined by the system designer. They can contain raw sensor data or more abstract information, e.g., from a user defined filter. The definition of measured characteristics should be well-considered since they define which effects of the real world are observable by the system.

In contrast to the measured characteristics, the number of **derived characteristics** C_d is not fixed. They can be generated by the system itself during runtime as a result

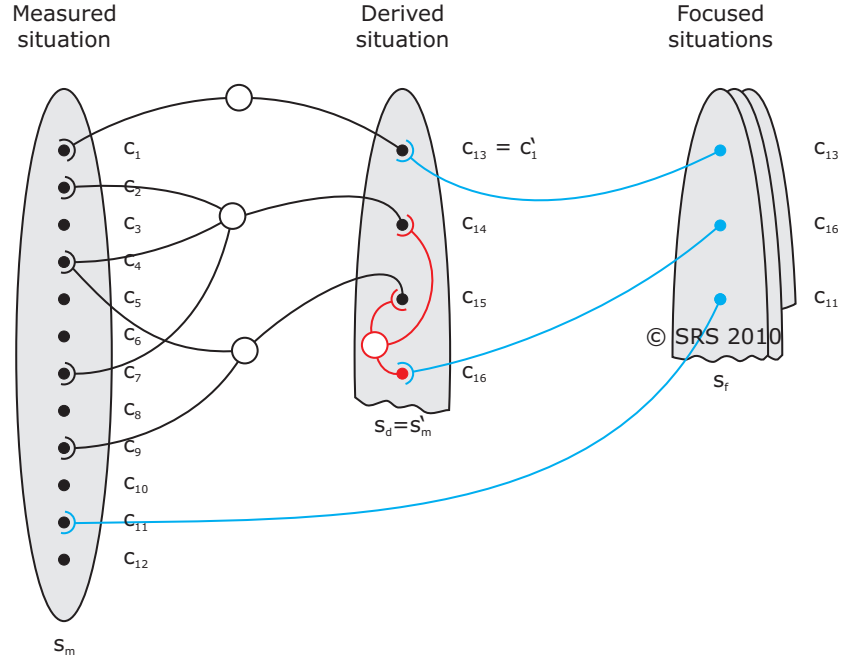


Figure 4.4.: Connections between the different types of situations

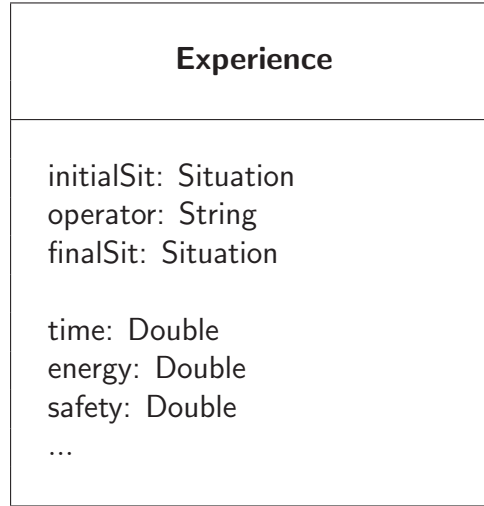
from learning. By the combination and filtering of measured characteristics and other derived characteristics, they build new virtual characteristics with a high degree of abstraction.

The set of **focused characteristics** C_f or the focused situation respectively is a variable combination of measured and/or derived characteristics. It can be changed completely from one situation to another depending on the current parameters of measured characteristics, derived characteristics, and the next operator(s) to be executed.

Different kinds of situations are also used in [Ahl07] distinguishing prefiltered and interpreted situations. Here, a prefiltered situation contains sensor measurements filtered by user-defined functions and algorithms respectively. Hence, it is similar to the measured situation. Interpreted situations are similar to the focused situations. Due to the different learning strategies proposed in this work, especially the automatic learning of relations which is not realized in [Ahl07], a level between measured and focused situation is necessary to keep the structure of the entire model clear and traceable.

4.1.2. Experiences and action spaces

The described situations represent time-fixed parts of the real world. In order to represent changes of the real world, two situations representing an initial scene and a final scene are required. The action(s) between these scenes are described by an operator or

Figure 4.5.: Class **Experience**

meta operator respectively. If the name of this operator and both situations are known, these three information can be taken together to describe the effect of a certain action regarding a certain initial scene.

According to [Söf01c], the combination of an initial situation s_{initial} , the name of an operator n_{operator} , and a final situation s_{final} is denoted as ‘experience’. In the proposed framework, experiences are represented by the class **Experience** (see Fig. 4.5) contained in the Petri Net as token. Besides the two situations and the name of the performed operator, also further information as the required time and energy for execution as well as an index for safety (or risk) of the related action or situation are connected to the experience object.

Several objects of the class **Experience** can be used to represent an action space if the final situations of some experiences are also initial situations of some others. If the objects are stored in a database or simple list, search algorithms or query functions can be used to estimate paths of operators between situations or to draw the action space. Hence, action spaces are represented by objects of the type `ArrayList<Experiences>` or a database object. Both objects are contained in the Petri Net as tokens.

In this thesis the open source object database db4o¹ is applied. The software can be integrated easily into own projects in order to store and to process object-oriented data structures (see [EHHP06, RV06]). The complete object hierarchy can be stored persistently and processed by three different query languages.

¹<http://www.db4o.com/> (retrieved on November 2nd, 2010)

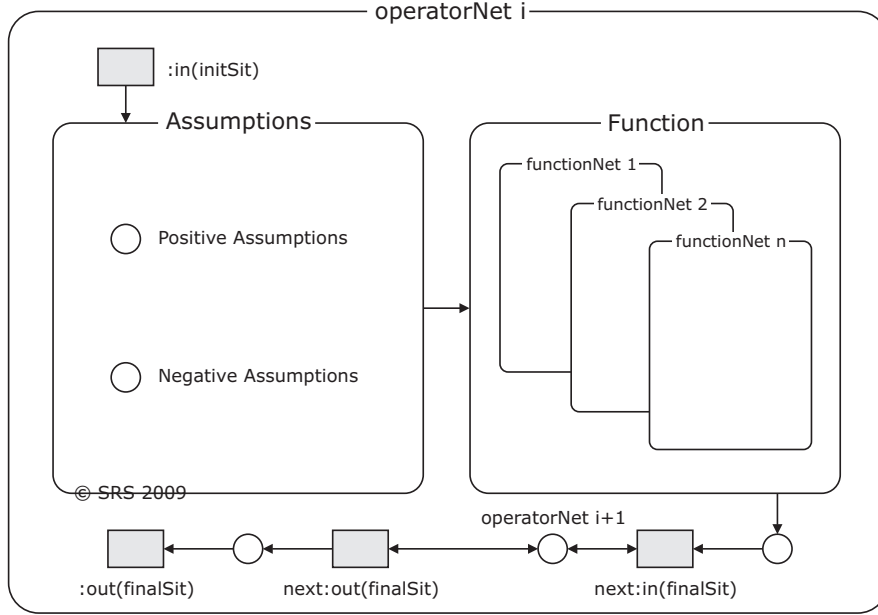
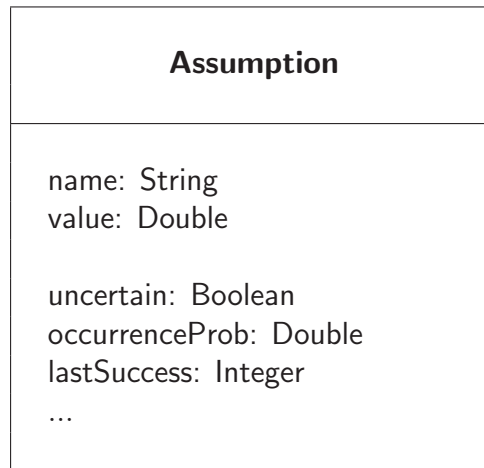


Figure 4.6.: Structure of an operator net

4.1.3. Passive and active operators

The SOM relations and operators are represented by special nets (operator nets). An operator net (see Fig. 4.6) gets an initial situation as input and generates a final situation depending on the operator net's function and assumptions. Several operator nets can be connected in a sequence (different to the term meta operator) to calculate the effects to the initial situation successively. The function and assumptions can be set during modeling as an initial mental model or during runtime to learn the interaction with the environment autonomously. Besides the representation as Petri Nets, also an equivalent representation by operator classes is used since this allows faster computation for complex systems and does not require a Petri Net simulator.

The assumptions of operator nets can be represented either positively or negatively by a 2-dimensional array of the class **Assumption**. Hence, the array consists of related individual lists of the class **Assumption**. Each list represents observed or inferred assumptions of the corresponding operator regarding a certain characteristic. A positive assumption is a condition which has to be fulfilled and a negative assumption is a condition which does not have to be fulfilled. Sometimes a single situation has to be set as an assumption and sometimes it has to be set as an exception or restriction. If the assumptions are not fulfilled, the operator net does not change a situation (initial situation is equal to final situation). The class **Assumption** (see Fig. 4.7) has the same quality as the class **Characteristic**. Furthermore, the class **Assumption** contains the three attributes **uncertain**, **occurrenceProb**, and **lastSuccess**. These attributes are used to represent uncertainty and they are detailed in the following section.

Figure 4.7.: Class **Assumption**

The function of an operator net is implemented by the combination of several ‘function nets’ changing the parameters of the characteristics. Hence, the whole function of an operator or the whole structure of a situation respectively is the combination of the functions of those operator nets with fulfilled assumptions. A function net contains an arbitrary function which outputs a list of the class **Characteristic** (characteristic list) and takes a list of parameters and two characteristic lists as inputs. One of the two characteristic lists is also used to parameterize the function and the other sets the characteristics for the output. The parameters of a function net could be simple values or even complex variables as models interpreting the sensor measurements, representing the environment, etc.

Operator nets can also be connected in parallel, e.g., two or more operator nets are related to the same characteristic and have the same function, but partially different assumptions. In this case, at least one set of assumptions has to be fulfilled. If the sequence of nets contains two or more operator nets which are related to the same characteristic and which have equivalent assumptions, but different functions, also several alternative final situations are generated.

Furthermore, the interaction with dynamically changing environments can be represented with the proposed approach. Therefore, the whole observable dynamics of the environment is considered as several ‘dynamic elements’ which are independent from each other and the ego system. In the real world, a dynamic element could be everything that acts independently from the ego system. The behavior of each dynamic element is represented by an operator net describing a waiting action of the ego system (waiting operator) and characteristics which are mainly influenced by the related dynamic element.

4.1.4. Uncertain knowledge and conflicts

In order to reduce the complexity of the real world, humans do not consider all sensed stimuli, but perceive their environment in an abstract manner. Furthermore, the attentional focus may change according to the current situation and context. However, this naturally implies that a situation or a problem can also be simplified to much. In psychology, this effect is known as ‘central reduction’ describing a behavior when humans reduce the dependency of a system to only one or a few central elements [Dör97]. Due to the fact that the proposed representational level assumes situations with different degrees of abstraction, a transferable effect can also be described for technical agents. In this regard, some conditions or effects can simply not be measured if the required sensors are not available or if the effects take place in an unobservable part of the environment (e.g., in another room). However, also if certain conditions or effects may be measured, they can only be taken into account if the corresponding characteristics are also included in the focused situation, which is considered for further information processing.

If a sequence of operator nets is built of experiences whose focused situations does not contain all characteristics that are necessary to describe the assumptions of a certain operator, it may represent the action logic ambiguously. In this case, the system observes two different effects of a certain action in two apparently equal situations. These ambiguities again can result in the fact that an action space derived from the operator net sequences (see Section 4.3.1) may contain paths which are mutually exclusive. Due to the fact that this uncertain representation offers different alternative paths to be selected by a planning function, the described case is denoted in the following as ‘conflict’ (see Def. 4.1).

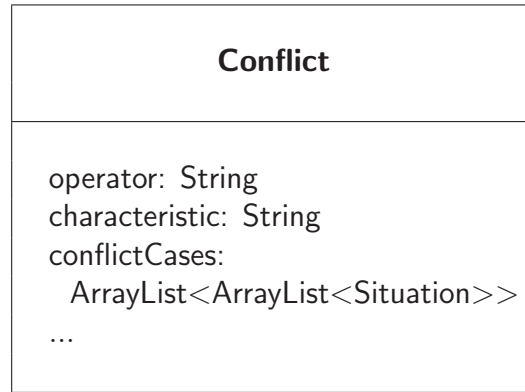
Definition 4.1: Conflict

An action space contains a conflict if one and the same operator transfers a certain initial situation to several different final situations.

In the following, two different types of representations for conflicts are presented. These representations are used to store further statistical and measured information about the existing conflicts gathered from interaction. The gathered information can be finally used to solve the conflicts.

Statistical information about conflicts

A conflict occurs if a sequence of operator nets already contains an operator net which has the same assumptions, but different functions, as a new operator net to be added. If this is the case, the corresponding assumptions can be marked as uncertain (attribute `uncertain=true`) and their occurrence probability (attribute `occurrenceProb`) is set to a value between 0 and 1. The occurrence probability λ_o describes how often a certain function connected to a certain set of assumptions was observed by the system related

Figure 4.8.: Class **Conflict**

to the entire number of functions connected to the same set of assumptions. A similar approach is also used in [Ahl07]. Here, a transition probability describes how often a certain operator was applied successfully to a certain initial situation resulting in a certain final situation. Besides the occurrence probability λ_o , also a variable μ_o is set (attribute **lastSuccess**), which describes when a certain assumption/function-pair was observed the last time instead of the other pairs involved in the same conflict. If a set of assumptions has a very low value of λ_o and a very high value of μ_o , the system can assume that the related operator net was resulting from an accidental observation which does not map the real world in general.

Measured information about conflicts

The representation of a conflict is realized by the class **Conflict**, which is generated if a conflict is detected the first time. The class **Conflict** (see Fig. 4.8) stores the performed operator, the changed characteristic, and a list of situations for each different function changing the characteristic. Furthermore, the common assumptions and different functions of the conflict cases are also stored as a link between the **Conflict** and the related operator nets. Whenever the conflict occurs, the initial situation of the experience is stored in the corresponding list of situations. Since the focused characteristics are not sufficient to solve the conflict, the stored situation gets all measured and derived characteristics of the system. If this situation is stored for each of the different functions (earliest after the second time a certain conflict is detected), the information stored in the related class can be used to determine the significant difference of the initial situations, which can be used to distinguish the different cases. Depending on the data types of the measured characteristics, different data mining methods can be applied to analyze the data sets stored in the class **Conflict**. Thus, the focused situation is extended by measured and/or derived characteristics (see Section 4.3.5).

4.2. Knowledge and cognitive functions

The proposed framework for Cognitive Technical Systems contains several models and modules, which are based on the described representational level. The models are used to store and exchange knowledge and the modules simulate cognitive functions by using and modifying the models. In order to extend this framework, new modules can be added easily if the defined representation is used to connect or combine them with the already existing modules. After the description of the models, the different modules simulating cognitive functions are explained. The framework consists of models for planning, perception, as well as goal generation and selection.

4.2.1. Knowledge representation

In the following, the developed models representing knowledge are explained. In contrast to previous work (see [Ahl07]), here, also the representation of perceptual knowledge besides the representation of action logic is considered. Furthermore, the hierarchization of these models is taken into account in order to reduce the complexity by focusing only the relevant parts of the real world.

Action model and action spaces

The actions of the ego system and the dynamics of its environment are represented in action models containing operator nets for each action (or sequence of actions as detailed below) of the ego system and each observed action of other independent agents. The action model can be generalized and specialized and it is a suitable description for long-term representation of action logic. However, due to the fact that the action model stores the conditions and effects of all actions, the calculation time for planning increases with the amount of stored information. Furthermore, the Petri Net representation is not suitable to derive a goal-directed sequence of actions directly. Hence, the focusing to a relevant part with a suitable format is also necessary.

In this thesis, action spaces are used to represent selected parts of the system's long-term knowledge. In order to generate an action space, the effects of all possible operator nets based on the current situation are calculated and stored as experiences. Then, the resulting situations are used as initial situations for the calculation until already explored situations result or if a certain number of experiences is reached. Hence, the action space may also change during runtime according to the current situation. This action space results from the combination of long-term memory (action model) and short-term memory (current situation) and can be utilized as working memory.

If an action model contains operator nets with uncertainty, also the resulting action space may be influenced. In such cases, operators are connected multiple times to the same initial situations and lead to different final situations (see Fig. 4.9). Hence, the

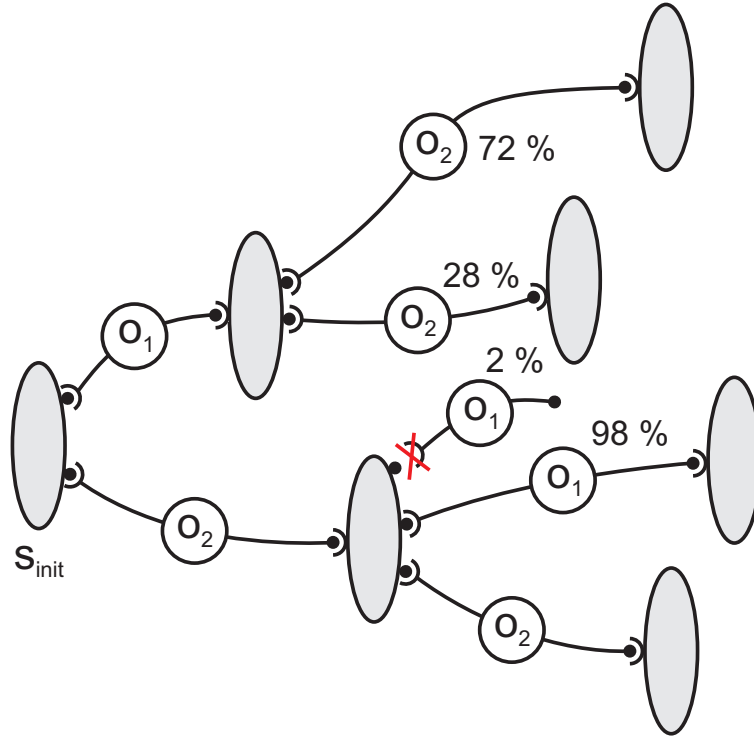


Figure 4.9.: Action space with alternatives

representation offers several action alternatives although only one actually exists. The resulting action space's operators are weighted with the occurrence probability leading to a representation that corresponds to a Markov chain (see [CN10]). Although the percentage weighting could be used for planning (see [Ahl07]), here, this stochastic representation is merely an indicator that the currently available knowledge is too general.

Hierarchical representation of action logic

In order to reduce the complexity of a real world's representation, a hierarchical structure is necessary. According to the proposed approach, human interaction can be divided into several action spaces differing with respect to the characteristics which have to be considered and the actions which can be performed. Hence, the result of human action-oriented reasoning is a hierarchical plan. The different steps of the plan are not detailed from the very beginning. They are detailed and modified dynamically if the human tries to reach the next sub goal.

A hierarchical representation with different degrees of abstraction can be realized by a structure of action spaces and meta action spaces with different orders. Here, the operators of meta action spaces are related to other action spaces or meta action spaces (see Section 3.1.3). However, in order to utilize the flexibility of this representation, which is

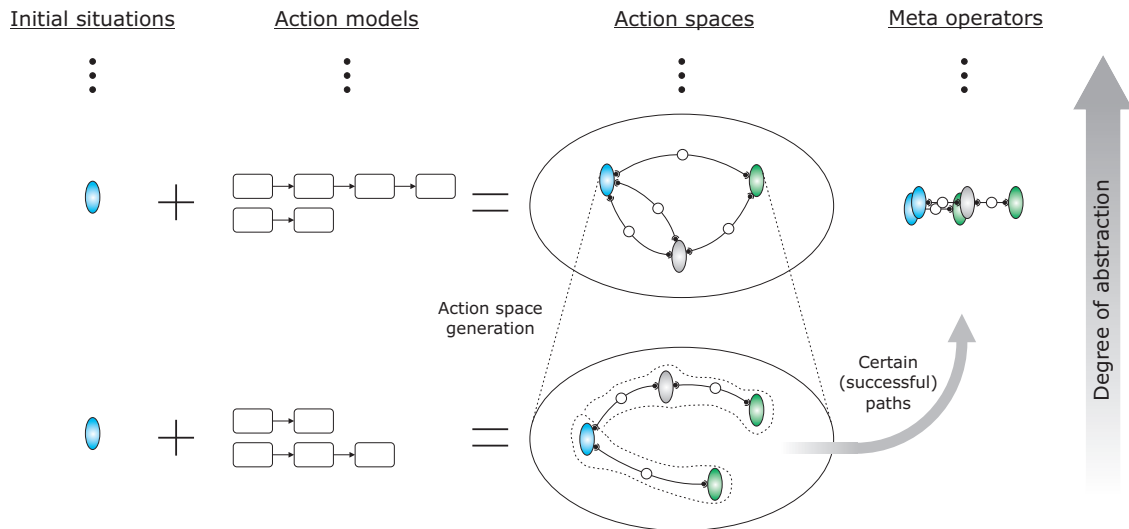


Figure 4.10.: Relations among action spaces, action models, and meta operators

restructured if the current situation and/or knowledge change, each action space has to be based on a generalizable action model. Thus, real world's action logic is represented by several action models with different degrees of abstraction. Each action model can be extended and refined by learning and it can be used to generate a context-sensitive action space with a corresponding degree of abstraction.

In Fig. 4.10, the relations among action spaces, action models, and meta operators are shown. The action space on the lower part of the figure describes a certain part of the possible real world's interaction and depends on the current situation and the criterion used to abstract the upper action space (e.g., the current goal). The operators of the lower action space are related to the basic actions of the considered systems and the lower action space itself represents a whole operator of the meta action space on the next higher level of abstraction. Furthermore, the operators of this meta action space may also be represented by meta operators. The assumptions and functions of each action space's operators are represented by the corresponding action models on the right side of the figure.

Meta operators

According to [Söf01c], a meta operator corresponds to a sequence of connected operators and is itself an operator on a higher hierarchical level. Its assumptions and function result from the combination of the connected operators' assumptions and functions. Thus, meta operators can be used to model actions on a higher degree of abstraction and describe strategies or rules to reach certain goal situations.

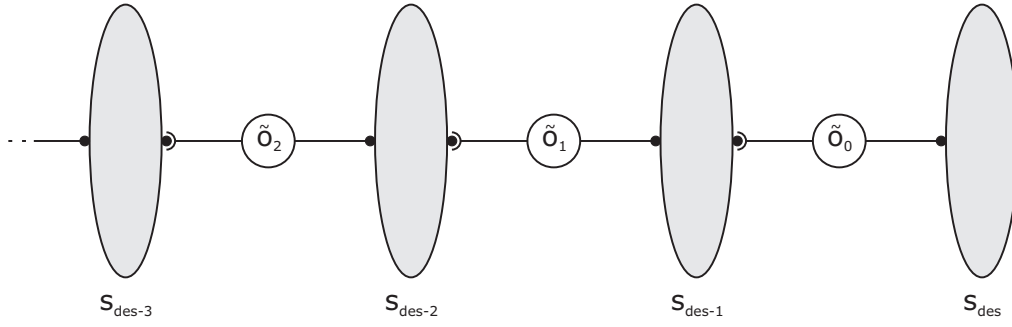
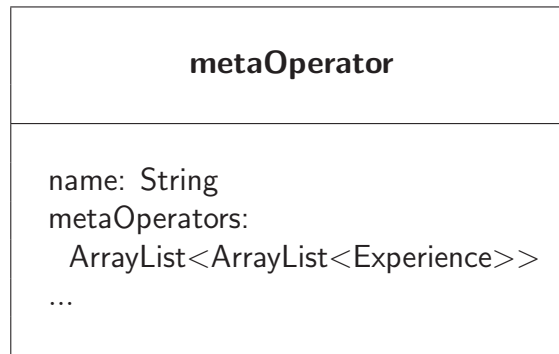


Figure 4.11.: Graphical representation of a meta operator

In this thesis, the real world's action-logic is represented by a hierarchical structure of action models and meta action models with different degrees of abstraction (see above). Here, the meta action models describe the general assumptions and functions of certain meta operators with the same degree of abstraction. If a plan is to be generated, the meta action model on the highest level is used to generate a meta action space. The contained operators are represented by a separate action model, which again is used to generate action spaces with a lower degree of abstraction. However, the generation of these action spaces stops if the situation resulting from the corresponding operator on the higher level is reached. The whole procedure repeats until a generated action space contains basic operators, which can be directly executed by a connected technical system.

However, besides the general representation of meta operators by the meta model's operator nets, the special effects of meta operators can be described by a sequence of experiences containing operators of the next lower level of abstraction (see Fig. 4.11). Hence, these sequences are rules, which describe how to achieve a certain final situation from a certain initial situation. The representation within the proposed framework is realized by a class `metaOperator` (see Fig. 4.12) storing the name of the meta operator and an array of the class `Experience`. The name of the meta operators is used to relate the class `metaOperator` to the operators of a meta action model. The class `metaOperator` is used to store executed sequences of actions which transferred the system to a goal and subgoal respectively (see also Section 4.3.3).

The representation of meta operators by sequences of experiences can be useful in different ways. First of all, a plan to be executed can be extracted from existing meta operators if they contain the current situation and goal. However, this is only applicable if the situations of the contained experiences consist exclusively of characteristics with nominal parameters. Independently, from the contained situations, a meta operator can also be used to support the planning process. If several different paths to the goal are identified in an action space which is not completely correct (may happen if the operator's assumptions were generalized), the meta operator provides the information whether one of the available sequences of operators in the path were already executed successively before. Hence, the system can prefer the previously executed path if the exploration of

Figure 4.12.: Class `metaOperator`

new (risky, but maybe faster) paths is not favored. Finally, meta operators can also be applied to speed up the generation of large action spaces. Here, the exploration of those paths is preferred which are also contained in meta operators leading to the current goal. If these paths do not exist, each path is considered equally. Hence, the action space generation alternates between an exploration based on breadth-first-search and depth-first-search.

Perception model

Some of the human's most powerful characteristics are the capabilities to recognize perceptual patterns and to switch the focus of attention to relevant aspects. Through these capabilities, humans consider the perceived world in an abstract manner depending on the current situation and context and they are able to handle the complexity of the real world. Hence, the implemented perception model is subdivided into a recognition and an attention model. The recognition model contains the relations which are used to determine the parameters of the derived characteristics and the attention model contains rules representing which characteristics have to be focused in which situation.

The recognition model contains operator nets from the same quality as the ones in the action model. These operator nets represent the relations and are linked to the situations by their assumptions. Due to the fact that the relations can not be measured directly, new operator nets can only be added through the designer in advance (human interprets structure of the situation) or through learning from experiences (application of pattern recognition methods). The operator nets stored in the recognition model are used during perception in order to determine the parameters of certain derived characteristics from the parameters of measured characteristics and/or other derived characteristics.

The attention model is realized by a set of rules describing which measured and/or derived characteristics are to be contained in the systems focused situation. This may depend on the next operator to be applied and/or a set of characteristics (e.g., related to the current goal or to a certain initial situation of the defined operator). Thus, the

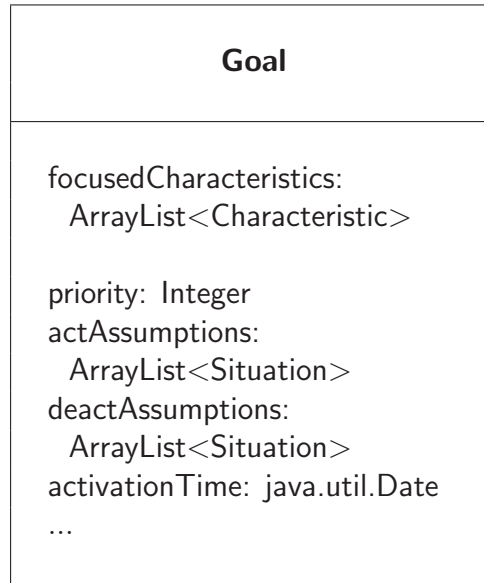


Figure 4.13.: Class Goal

focused situation can be set dynamically during perception and is a context-sensitive representation of the current scene and its mental mapping respectively. As the recognition model, also the attention model can be set by the system designer and also refined automatically from interaction with the environment.

The learning of new relations and new characteristics to be considered is one of the main contribution presented in this thesis. Thus, the situation's internal structure can be adapted to a certain environment automatically in order to realize autonomous systems performing independently from their initial knowledge. The related learning mechanisms are detailed in Section 4.3.5.

4.2.2. Goal generation and selection

The behaviors of biological systems as humans and animals are strongly influenced by natural drives. These drives can be divided into primary drives that exist after birth and secondary drives which are developed in the first years of a being's life (e.g., see [Hul52]). Examples for primary drives are the sex drive or hunger assuring the surviving of an individual or a whole population. Social acceptance is an example for a secondary drive. From a black-box point of view, drives increase with time or certain circumstances, which is inspiration for the generation and selection of goals in the proposed framework.

A goal usually denotes a desired state or situation and the interaction of a considered system is characterized in order to transfer the system from the current state to the desired one. Furthermore, usually several goals (also those which exclude each other) with

different subjective priorities exist. In the proposed framework, a goal is represented by the class **Goal** (see Fig. 4.13) and contains a list of characteristics such as the class **Situation**. Hence, the system has to find a sequence of actions leading to a situation whose characteristics have the same parameters as the goal's characteristics.

In contrast to the class **Situation**, the class **Goal** has additional attributes which are modified and used by a goal generation and a goal selection module. The additional parameters are

- priority,
- activation assumptions,
- deactivation assumptions, and
- activation time.

The priority is an integer value between one and ten. The lower the value of priority, the more urgent is the goal. The activation and deactivation assumptions are lists of situations. If a situation in one of the lists is a subsituation of the current situation, the current goal is activated and deactivated respectively. Finally, the activation time simply stores when a goal was activated.

In the framework, goals are either active or inactive and stored in one of two related lists (see Fig. 4.14). The goal generation module compares the current situation with the activation and deactivation assumptions of each goal and shifts the goal to the corresponding list if required. If a goal is shifted from the list with inactive goals to the list with active goals, also the activation time is updated. The characteristics of the goals and the situations for activation and deactivation are defined by the system designer. However, goals could also be learned from interaction by generating new goals and storing them in one of the two lists.

In order to select a goal from several ones, a goal selection module is used. This module compares the priorities of the goals in the active list and selects the goal with the highest priority. If two goals have the same priority, the activation time can be used to select the goal which was activated firstly. Alternatively, the goal selection module could also take the current action space into account in order to check which goal can be achieved faster. Furthermore, the whole set of goals and the current action space could be used to generate a complex strategy in order to achieve all goals in an optimal sequence regarding time or energy.

4.2.3. Planning

As described in Section 2.2.1, *planning is the reasoning side of acting* (see [GNT04]). Accordingly, the interaction with the real world is mentally simulated a priori in order

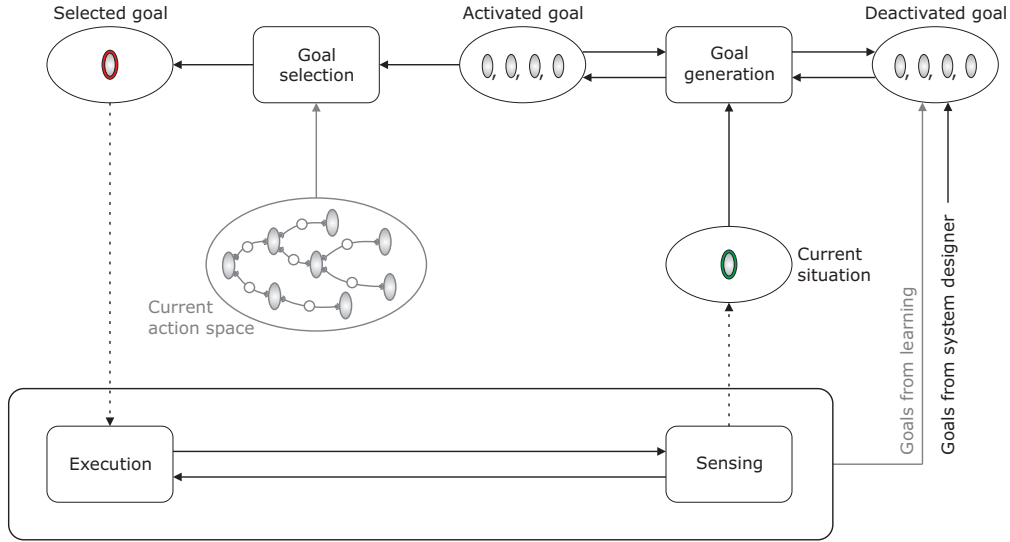


Figure 4.14.: Goal generation and goal selection

to realize goal-directed behavior. Furthermore, this also implies the existence of a representational level.

In the proposed approach, the real world is mapped internally by a combination of different kinds of representations. However, here, the action space is best suited for planning since it is a reduced and context-sensitive derivation of the other representations. More precisely, the action space contains all known and currently available action alternatives regarding a certain degree of abstraction.

The considered action space is analyzed by a planning module to generate an appropriate list of operators whose sequential application would transfer the system from the current to a desired situation. Since the action space is a state space representation, search algorithms can be applied. If the situations or operators in the action space are not weighted, breadth-first-search may be applied to generate a path with a minimum number of actions. If situations or operators are weighted, the Dijkstra algorithm [Dij59] can be applied. Furthermore, also other search or optimization algorithms can be added simply into the planning module.

In order to use the planning module a certain search algorithm has to be set. Then, the module takes a set of experiences (here: an action space), the current situation, and a goal situation to generate the plan. The resulting plan consists of a sorted sequence of experiences where the initial situation of the first experience is the current situation and the final situation of the last experience is the goal situation. The situations are contained in the plan in order to compare them with the situations which are obtained if the plan is executed in the real world.

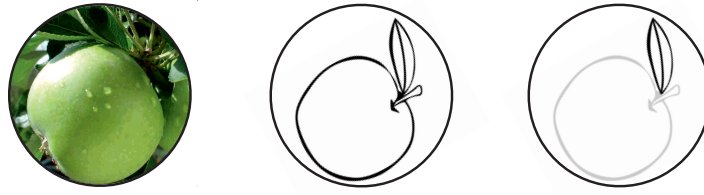


Figure 4.15.: Different representations of an apple

4.2.4. Perception

A cognitive system's function of perception is more than the sensing of physical quantities. Perception implies also the fusion and interpretation of stimuli from different sources by the capability of pattern recognition. Furthermore, only a small relevant part of the sensed data is really focused, which is typically denoted as attention. Hence, the complex scene of the real world is mapped to an abstract and context-sensitive situation.

As an example in Fig. 4.15, three different representations of an apple are illustrated. The left representation shows a photo of a green apple. The photo consists of a large number of colored pixels. These pixels are sensed by the human's eye, but they are not perceived by the human brain completely. The sensed data are directly interpreted in order to derive more abstract information as illustrated by the middle representation. Here, a large amount of data is reduced to the necessary information and some irrelevant attributes as the color or texture of the apple (depends on the context) are not considered. Finally, the right representation illustrates the attentional focus on a certain part of the abstract representation (here, the leaf of the apple).

In order to simulate the cognitive functions of recognition and attention, the three different kinds of situations (see Section 4.1.1) are processed by two related modules which are combined in a perception module. The recognition module determines the parameters of derived characteristics from measured characteristics and other derived characteristics. After that, the focused situation is established by the attention module selecting certain characteristics from the measured and derived characteristics.

Recognition module

The recognition module takes the measured characteristics (which are closely related to the physical quantities sensed by the system) and determines the (usually nominal) parameters of derived characteristics. These derived characteristics can also be taken to determine the parameters of other derived characteristics and so forth. Hence, a hierarchical structure with different degrees of abstraction can be established. The SOM relations among the different characteristics are represented by the sequences of opera-

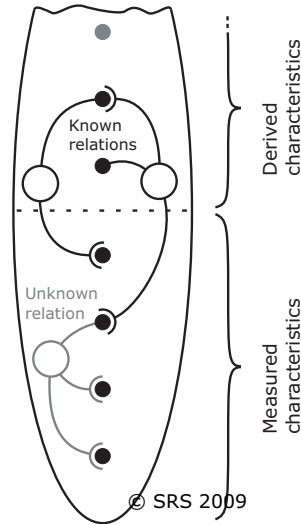


Figure 4.16.: Measured and derived characteristics in focused situation

tor nets (and corresponding Java objects respectively) stored in the recognition model. Here, each sequence of operator nets corresponds to a certain unique derived characteristic. Whether these relations may be applied or not (the parameter of the corresponding derived characteristic is changed or not) is represented by their assumptions (similar to the assumptions of active operators). The final result is a set of measured and derived characteristics, which is further processed by the attention module. A derived characteristic abstracts information of another characteristic or fuses the information of several other characteristics. Hence, the described recognition module enables the system to reduce the measured scene and handle the complexity of the real world.

Attention module

The attention module establishes the focused situation (see Fig. 4.16) based on the available measured and derived characteristics. Therefore, the rules stored in the attention model (see Section 3.1.2) are used to select those characteristics which are relevant in the current context. Here, the current context is related to the actual goal to be reached and/or the next operator to be applied. Moreover, the subsituations resulting from attention (and recognition), which are reduced representations of the measured scenes, can be used to generate meta actions spaces with different degrees of abstraction (see Section 3.1.3).

4.3. Key feature: learning

Autonomous systems are in one way or another independent from their initial knowledge. Hence, an autonomous system has to be able to learn from interaction due to the fact that the environment is usually not static and partially unknown. Cognitive systems have this ability. They acquire new knowledge from interaction and refine/re-structure their mental models of the real world by a variety of different mechanisms leading to a better performance in future interaction. Here, also hierarchical abstraction plays an important role to detect patterns and to reduce the complexity of gathered data. Accordingly, the realization of autonomy requires more than the simple storage of input/output-pairs or the adaption of certain parameters and the interplay among different cognitive functions has to be considered.

The proposed framework supports several different learning mechanisms which can be related to three different categories regarding an alternating execution of perception, planning, and interaction. Here, learning takes place during interaction, after interaction by perception, as well as before or after planning (see Fig. 4.17). During interaction perceptual skills and skills for action control are tuned and optimized. After an action or a sequence of actions is executed, the perceived effects are stored in the short-term memory, integrated in the action models, stored as meta operators, and/or used to

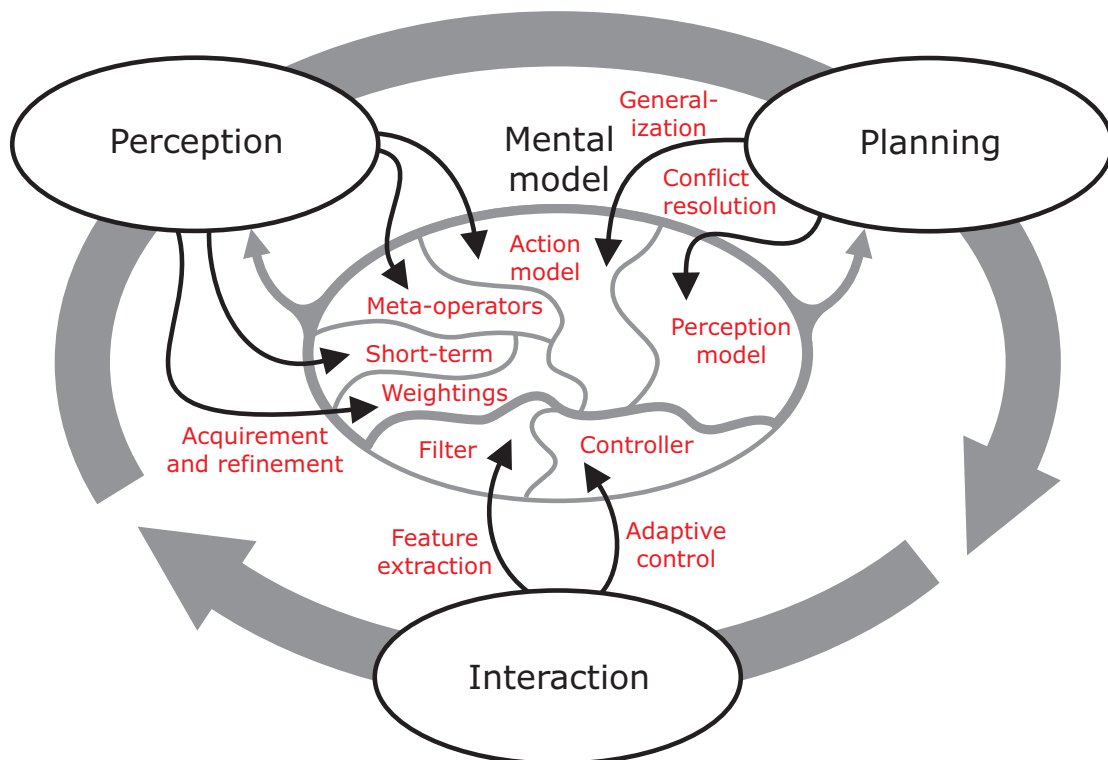


Figure 4.17.: Different learning functions

weight operators. Also during planning, where explicit represented knowledge is used, the system learns by generalizing or specializing the old knowledge.

In the following, the supported learning mechanisms are described in detail. At first, the learning of action models from experiences, the weighting of operators, and the learning of meta operators are presented. Afterwards, the storage and usage of perceived experiences in a short-term memory is detailed. Finally, the learning during planning which is generalization of operator's assumptions and specialization through conflict resolution is presented.

4.3.1. Operator's function and assumptions

The functions and assumptions of operators are represented by experiences and operator nets as well. Both representations have certain advantages and are useful for different purposes. An experience represents the effect of a certain operator related to a certain initial situation and several experiences can be used to represent a whole action space. The action space considers the relevant interaction based on the current situation and can be utilized for planning. In contrast to that, operator nets represent the action logic in a general manner. The action models can have different degrees of abstraction and the assumptions of the contained operator nets can be refined by deduction and induction in order to learn a general representation of an operator through interaction.

In order to use the advantages of both representations, they have to be converted into each other. In Fig. 4.18, the learning of an action model from measured experiences is illustrated. After a planned action was executed and the perceived effects were stored as experience, an operator net is created for each characteristic of that experience. The new operator nets get the characteristics of the initial situation as assumptions and the function is derived from the difference between the related characteristic in the initial and final situation. In the easiest case, the function is a simple change from the characteristic's parameter in the initial situation to the characteristic's parameter in the final situation. The new operator nets are integrated in the existing operator net sequences of the action model if they are not contained so far. Here, also conflicts can occur (see Section 4.1.4), whose solution is described in Section 4.3.5. From the action model and the current situation, a new action space consisting of experiences is generated. The action space again is used to plan the next action to be executed.

In order to learn more complex functions of operators, the action model has to contain a certain number of operator nets whose functions have a similar structure. For example, if a certain operator adds or subtracts a certain value to/from a certain characteristics, several operator nets with the function net **change** can be replaced by one operator net with the function net **add** which requires a positive or negative parameter k added to the parameter of the initial characteristic. Function nets with a more complex transfer behavior, e.g., with a high order differential equation, are possible, but not suitable since the proposed approach assumes symbolic representation for the action logic. The

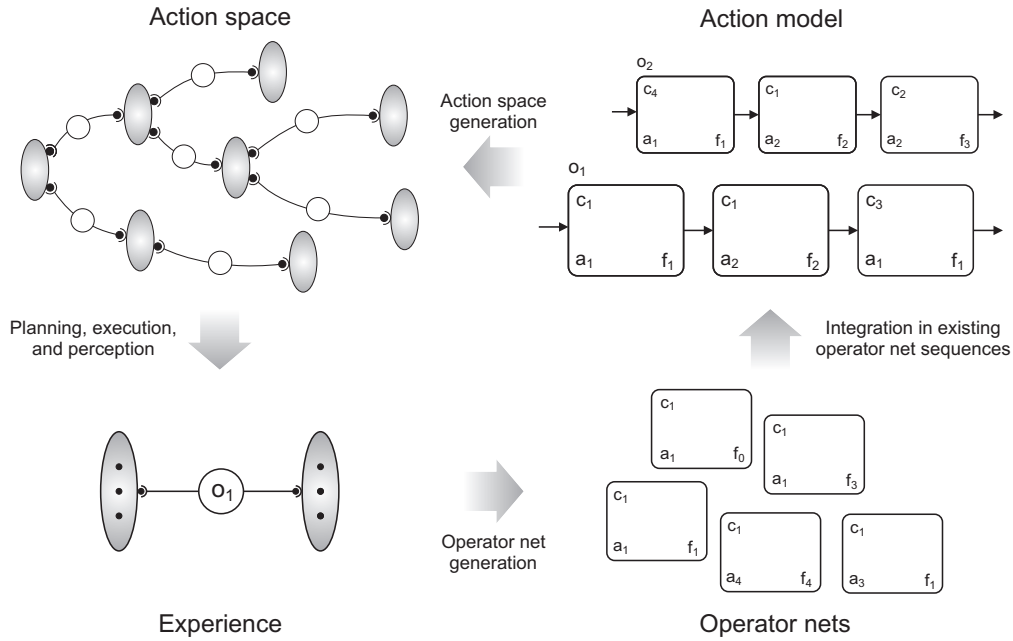


Figure 4.18.: Learning of operators' functions and assumptions

subsymbolic representation is realized by perception, where nominal values are derived from numerical values.

The deviation of experiences from the action model is related to the systems reasoning capabilities. A detailed description of this process can be found in Section 4.2.1.

4.3.2. Weighting of operators

Especially for planning, several different operators have to be compared in order to calculate the best path according to a certain criterion. For example, the time for executing an operator successfully could be such a criterion. Although the needed time could be described by an operator's function (e.g., represented by an initial and final situation), this should be avoided due to several reasons. If time would be represented by characteristics, the estimation of general assumptions and functions would be very difficult since time continuously increase. Additionally, the time to execute an operator may vary extremely depending on the specific current situation.

Nevertheless, criteria as the required time should be considered in a certain manner. Hence, further variables are required to describe common qualities which are important for each operator independently from the operator's assumptions and functions. However, to avoid the storage of thousands of different experiences with different time values, statistical quantities as the mean value or the variance can be used.

Due to the fact that criteria as the time is related to a certain situation in which the operator is applied, the weightings of operators can be represented by experiences. Since the experience is represented as a class, further parameters can be simply added as attributes. Several experiences regarding a certain operator can be stored in a list (one list for each operator). If an action space is generated, the calculated experiences can be compared with the lists of previous executed operators in order to assign the corresponding weightings to the action space. Furthermore, the initial situations of the stored experiences can also be generalized if the experiences have the same weightings.

Besides the mean value and variance of needed time, also other values of statistical variables can be derived after the execution of an operator and used for future planning processes. Examples for these variables are

- the average energy consumption,
- the degree of an operator's safety or risk, and
- the uncertainty of knowledge.

As the required time, the required energy for the execution of an operator can also be simply measured, e.g., by comparing the charging states of batteries before and after execution. The calculation of safety or risk may be estimated by storing the ratio of trials an operator was successfully executed (or stopped if a certain characteristic exceeds a critical value etc.). Furthermore, the safety of operators and situations respectively can be derived from a model with general safety principles, which may be set by the designer (see [EGVS10]). Finally, the statistical variables can also be used in order to describe the uncertainty of knowledge in addition to the representation given in Section 4.1.4

The discussed further quantities are merely statistical values. They are neither suitable to describe action logic nor to describe the internal structure of situations. The quantities can be simply logged during interaction and used for planning in order to select a certain path from several alternatives.

4.3.3. Meta operators

The assumptions and function of a meta operator can be represented generally by a meta action model or more specially by a sequence of experiences (see also 4.2.1). Both representations are closely related to each other and can be learned from interaction with the environment. By meta action models, the action logic is hierarchically structured and represented in a reduced and context-sensitive manner. Furthermore, knowledge about previously successful sequences of experiences may support or supersede planning processes in the future.

Meta action models may be derived from an existing action model with a lower degree of abstraction. From the existing action model an action space is generated. After that,

the initial and final situations of the action space's experiences are reduced to subsituations. Additionally, these subsituations can also be extended by characteristics which are derived from the characteristics contained in the original situations. After that, the generalized experiences are used to build the meta action model. As a reference for the characteristics remaining after generalization, the system's goals (subsituations of the focused situation) should be taken into account. Hence, it is guaranteed, that the achieved meta actions spaces can be applied for planning.

For each meta operator which is identified if the meta action model is derived, a new class `metaOperator` is generated. This class is extended by operator sequences which are learned from interaction. Here, especially successful plans are stored since they describe how to transfer the system through an action space from one subgoal to another. If the system observes several different sequences of experiences which correspond to the same meta operator, these sequences are stored also in the same class.

In [Ahl07], meta operators are also learned from interaction. They are derived from successful plans and from operator sequences in the database, which are free of ambiguities. Here, a meta operator is a sequence of several basic operators. Its assumptions correspond to the assumptions of the first operator and its function results from the sequential execution of all basic operators. The meta operators are used to reduce the size of the database and to speed up the planning process. Similar approaches are also used in other cognitive architectures. For example, in ACT-R 'production compilation' is used to generate a new rule from a sequence of others, which has the same function as the original rules [ABB⁺04]. Through production compilation, rules are specialized. However, depending on the rules, this approach can also be used for the generalization of knowledge [TD03].

4.3.4. Short-term memory

If the action space contains conflicts (operators whose functions are mutually exclusive), the correct path matching to the real world can not be estimated by weightings and meta operators.

As described in section 4.1.4, a conflict occurs if the system observes different effects of an operator although the operator is executed in apparently equal initial situations. This is the case if the environment is changed permanently or temporally and also if the system does not consider the relevant details of a scene (see also subsection 6.5). Due to the fact that the system stores the successfully execution of an operator in the corresponding assumptions' attributes `lastSuccess` and `occurrenceProb` (see section 4.1.3), permanent changes of the environment can be easily detected. However, if the environment or the behavior of other agents changes temporally, the described uncertainty remains during the whole interaction.

Here, a short-term memory may be helpful in order to remember which paths of the action space could not be followed and in order to avoid recurrently planning of plans which are not successful. By using a short-term memory, the system is able to solve current conflicts by taking the recent past into account (e.g., an operator will effect the real world in the same way as observed a few minutes ago).

The short-term memory is represented by a list of experiences which is currently updated by the newest experience (derived from executed operators and meta operators) assigned with a time stamp. In order to define how long the experiences are stored in the short-term memory a global dwell time has to be set. If the difference between an experience's time stamp and the current time is larger than the dwell time, the experience is deleted from the list.

If the mental action space contains conflicts (an operator leads to different final situations although it is connected to the same initial situation), the short-term memory is applied to check whether one of the alternative functions was recently observed. In this case, the alternative function (represented as experience in the mental action space) corresponding to the recently observed one is considered as the valid one and the other alternative functions are ignored or deleted from the mental action space. If more than one alternative functions (related to the same conflict) are stored in the short-term memory, only the latest one is considered.

4.3.5. Situation's structure

The learning mechanisms described in the sections above are used to build a representation of the real world which can be used for action planning. During these processes, the function of perception is used continuously to reduce the complexity of sensed measurements and mental representations as well. However, due to the fact that the real world may change, also the system's perception capabilities have to be adapted.

In the proposed framework, the cognitive function of perception is subdivided into recognition and attention, which make use of the system's knowledge about the situation's structure. In order to refine this knowledge, three different learning mechanisms, which realizes processes of deduction and induction, are presented in the following. These learning mechanisms

- generalize the operators' assumptions,
- add new characteristics to perceptual focus, and
- generate new relations among the characteristics.



Figure 4.19.: Robot learning the assumptions of the operator $O_{\text{driveForward}}$

Thus, a system becomes capable to generate an abstract and focused view of the real world automatically. This again reduces a system's dependency on its initial knowledge implying a high degree of autonomy.

Example scenario

In the example scenario, a mobile robot has to drive forward by increasing its translational velocity (see Fig. 4.19). However, in order to avoid a collision with any kind of obstacles, the application of the operator $O_{\text{driveForward}}$ is only possible if the distance between the robot and an object in front of the robot is larger than 500 mm. Since this distance is measured with a laser range finder which can only detect objects with a minimum height of 300 mm, also two light barriers inside the robot's gripper are used to detect smaller objects. Hence, the operator $O_{\text{driveForward}}$ can also not be applied if one of the two light barriers within the gripper is closed. In both cases (distance too short or light barrier closed), the motors of the robot are blocked in the corresponding direction.

In the scenario, the robot's measurements are represented by ten different characteristics and the initial and final situations are stored as experience after the operator $O_{\text{driveForward}}$ is executed. The ten characteristics are

- C_{transVel} {zero, slow, medium, fast},
- $C_{\text{redObjDet}}$ {true, false},
- $C_{\text{greenObjDet}}$ {true, false},

- $c_{\text{yellowObjDet}} \{ \text{true}, \text{false} \},$
- $c_{\text{blueObjDet}} \{ \text{true}, \text{false} \},$
- $c_{\text{x-position}} \{ p \in \mathbb{R} \},$
- $c_{\text{y-position}} \{ p \in \mathbb{R} \},$
- $c_{\text{orientation}} \{ p \in \mathbb{R} \mid 0 \leq p < 360 \},$
- $c_{\text{distanceToFront}} \{ p \in \mathbb{R} \},$ and
- $c_{\text{lightBarrierClosed}} \{ \text{true}, \text{false} \},$

whereas only the characteristics $c_{\text{distanceToFront}}$ and $c_{\text{lightBarrierClosed}}$ are required to estimate whether the operator $o_{\text{driveForward}}$ can be executed. If the operator $o_{\text{driveForward}}$ is executed the parameter of the characteristic c_{transVel} changes from **zero** to **slow**, from **slow** to **medium**, or from **medium** to **fast**. However, in this example only the change from **zero** to **slow** is considered. The set of situations representing the assumptions to be generalized are generated randomly in order to have a representative number of different examples. At first a table of 1000 different situations is generated, whereof only 25 % fulfill the assumptions of the operator $o_{\text{driveForward}}$. Then, the table is permuted to calculate 50 different sequences of situations. However, only the first 100 situations are considered.

Generalization of assumptions

The operator's assumptions are learned regarding the focused situations. Hence, the initially learned sets of assumptions are usually too special and some of the contained characteristics might not be relevant to describe whether an operator can be applied or not. A possible reason for that are changes in the environment, but also simplifications and mistakes in the initial knowledge (set by the system designer). The irrelevant characteristics have to be estimated and deleted from the operator's set of assumptions. Thus, the assumptions are generalized and the operator can be applied mentally also to those situations which were not explored before in the physical world (see also [GS10b, GS09]).

In order to reduce the assumptions of an operator to the relevant ones, a regression analysis can be used. Therefore, all assumptions of the operator nets changing a considered characteristic c_i with a certain function F_i and all assumptions of the operator nets changing the considered characteristic c_i in another way (or not) are analyzed regarding correlation. If some characteristics correlating to the function F_i are found, the correlating characteristics can be used as a new set of assumptions replacing the old sets of assumptions of the corresponding operator net.

As an example, the described robot scenario is used. All six characteristics with nominal values are contained in the focused situation. Furthermore, the characteristic

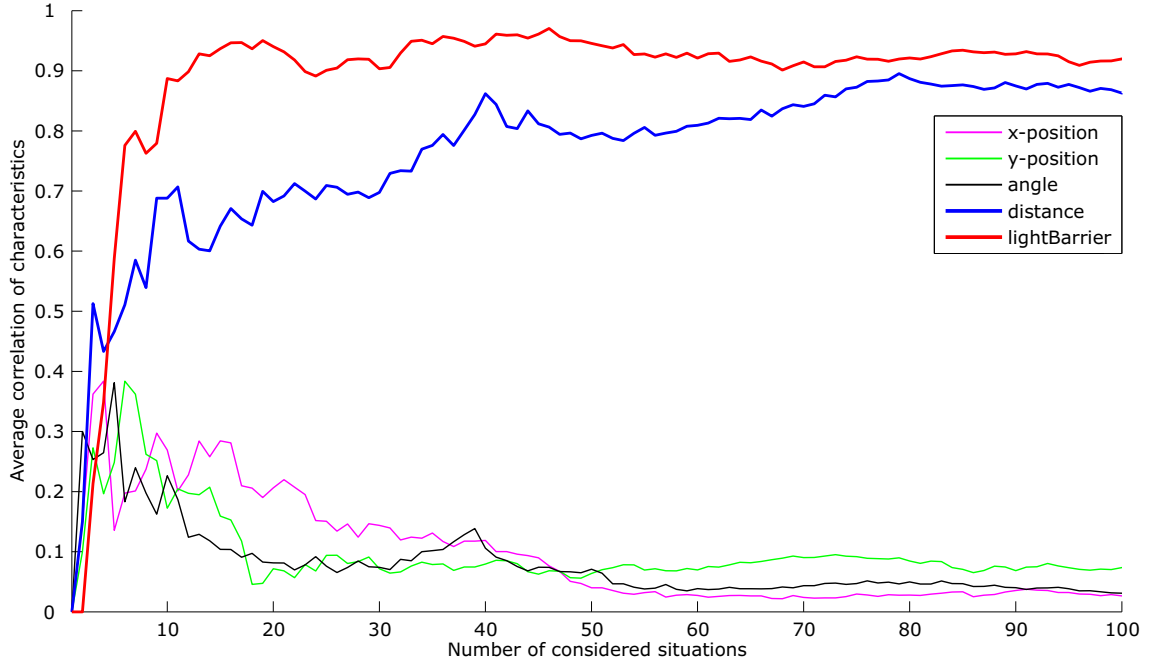


Figure 4.20.: Correlation of the general assumptions in the example scenario

$c_{\text{distanceToFront}}$ is converted to a Boolean value and added to the focused situation. If the distance is smaller than 500 mm the parameter is **false** and if the distance is larger or equal than 500 mm the parameter is **true**. Hence, the focused situation and the set of learned assumptions as well consist of six characteristics with nominal parameters. Several focused situations and the information whether they fulfill the operator's assumptions are generated randomly and used in an accumulated way to estimate which characteristics correlate. If a characteristic correlates, a related weighting increases and vice versa.

In Fig. 4.20, the average correlation of the six characteristics over the number of considered example situations is shown. The diagram shows that the weightings of the relevant characteristic increase and the weightings of the irrelevant characteristics decrease significantly if only 5 to 10 example situations are considered. This difference gets larger with increasing number of considered example situations.

Conflict resolution and learning of new characteristics

The learned action model has a certain degree of uncertainty and the corresponding action space based on this action model contains operators with alternative functions if a focused situation is too general (characteristics which are crucial for action planning are missing). Hence, conflicts exist and the system has to decide among several alternatives which may be true or not. Here, the decision could be realized simply by a selection based on the occurrence probability λ_o (see Section 4.1.4). However, this

heuristic approach is not applicable for systems in dynamical environments since it is strongly influenced by random occurrences and may never guarantee a correct solution (also not if the environment does not change). In contrast to that, the existence of uncertain knowledge should initiate a process of refining the mental model and to solve the conflicts respectively (see also [GS10a]).

As described in Section 4.1.4, conflicts occur if the mental model has not enough information or is not detailed enough to determine the differences between apparently equal situations. In order to solve this problem, it can be helpful to extend or restructure the focused situation which is considered if the operator (with the uncertain representation) is executed. Therefore, additional information (here: additional measurements) have to be gathered to estimate one or several new characteristics solving the conflict.

The new characteristics can be selected from the set of measured characteristics which are stored in corresponding class **Conflict**. In order to select the new characteristics solving a conflict, a regression analysis can be applied. Therefore, all situations stored in the lists of the class **Conflict** build the set of examples with the different cases as labels. Depending on the result of the analysis, all attributes of the example set, which correspond to the measured characteristics, get weightings. The weighting is used to judge whether a measured characteristic is added to the considered characteristics or not. How many characteristics are added and/or how large the threshold weighting has to be, can be defined separately.

In the example, it is assumed that the operator nets describing the function of the operator `o_driveForward` contain only the characteristic `c_transVel` as assumption. Hence, the system assumes that the operator can be applied in every situation where the robot does not drive forward. The other nine characteristics are contained in the measured situation. According to this measured situation with nine characteristics, example situations are generated randomly. These random situations and the information whether they fulfilled the assumptions of the operator `o_driveForward` are used sequentially to determine which characteristics correlate to the change of the characteristic `c_transVel`. The results of the example simulation (see Fig. 4.21) are similar to the results of the previous example (see Fig. 4.20) with the difference that some characteristics also have numerical parameters. The weightings of the relevant characteristics increase significantly if five to ten example situations are considered.

The class **Conflict** keeps unchanged if no correlating characteristics are found (e.g., due to a lack of sufficiently different example situations). In this case, the conflict is tried to be solved again when it occurs the next time. If correlating characteristics are found, the characteristics are added with corresponding parameters to the operators function. Furthermore, the related characteristics are added to the focuses characteristics and the class **Conflict** is marked as not active. Now, the characteristics in the focused situation should be sufficient to determine the operator's correct function related to a certain situation. However, the simple adding of situations is only recommended for charac-

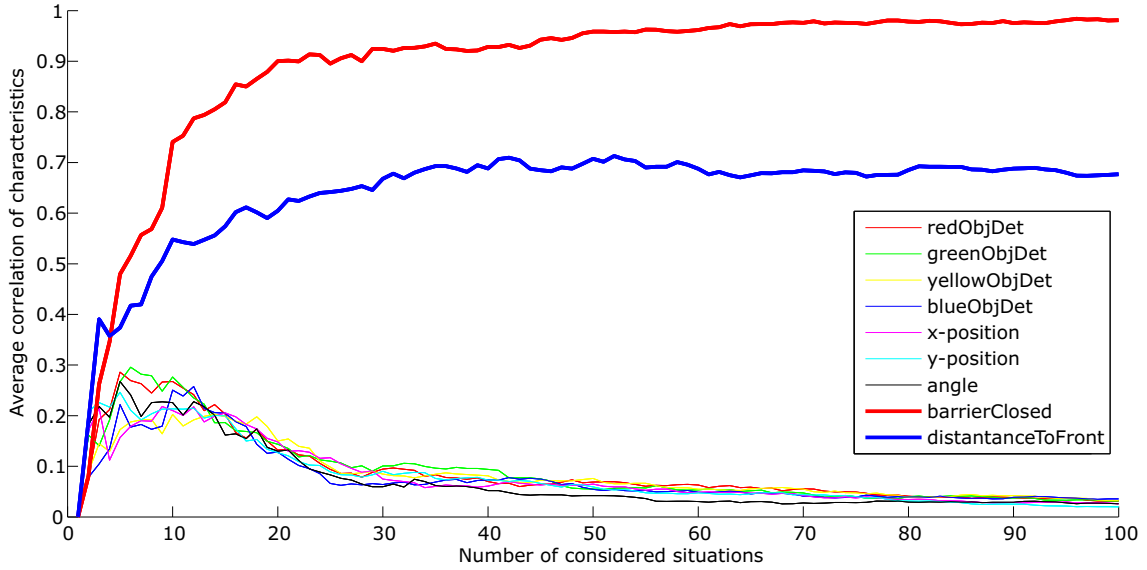


Figure 4.21.: Correlation of the characteristics required for conflict resolution in the example scenario

teristics with nominal values. If a correlating characteristic has a numerical parameter, the characteristic has to be converted to a nominal value which is added to the focused situation instead of the correlating characteristic. Through this process, a new relation is generated, which is described in the following section.

In the case that a deactivated conflict occurs again, the class **Conflict** can be reactivated. In this case, the result of the analysis was not detailed enough, usually due to a lack of examples. Through the reactivation, the old and new data can be mixed to get a better result. Furthermore, after deactivation of a class **Conflict**, the added characteristics can also be stored. Hence, the extension of the focused situation can also be reversed to keep the focused situation as small as possible. Furthermore, characteristics which are added to the focused characteristics wrongly can also be removed by a future generalization process.

Learning of new relations

Characteristics with numerical parameters are not suitable as assumptions without further interpretation since a certain numerical parameter usually does not occur several times again. Furthermore, the deviation of an operator's function regarding characteristics with numerical parameters may be relatively complex. Hence, the focused situation should only contain characteristics with nominal parameters (although also characteristics with numerical parameters are possible). In order to consider the corresponding physical quantity of a characteristic with a numerical parameter even so, the characteristic has to be converted to a characteristic with nominal value. In this case, the

two characteristics are linked via a relation, which could be described by the system designer. However, the anticipation of all situations occurring in a dynamical system, which should be moreover free of mistakes, can be a very complex task. Accordingly, a technical system should also be able to learn new relations from interaction with the environment automatically.

New relations can be learned by analyzing the coherence between an operator's function and the initial situation. Hence, the examples to be analyzed consist of the initial situation's characteristics and an additional attribute labelling different functions of the operator (e.g., label = true: parameter of characteristic c_1 changes from 1 to 3, label = false: parameter of characteristic c_1 does not change). The result of this analysis is a model which represents the relation between the characteristics contained in the initial situations and a new characteristic with nominal parameter, which is related to the previous label attribute. Hence, the model can be used to derive the parameter of the new characteristic from the parameters of the characteristics contained in the initial situations.

This described procedure can be applied to generalize assumptions or solve conflicts. If an operator's assumptions should be generalized the set of examples to be analyzed consists of the observed special assumptions. After a successful analysis, the previous assumptions are exchanged by the new characteristic derived by the model. In the case of conflict solution, the set of examples to be analyzed consists of the gathered measured situations which are stored in the class **Conflict**. If an unambiguous model could be generated, the new characteristic can be added to the focused situation to avoid conflicts in the future. Alternatively, the procedure can be applied after a regression analysis. Here, considered example situations contain only the determined correlating characteristics in order to convert them to a characteristic with nominal parameter.

The learned model is stored as the function of a new operator net which is added to the recognition module. Hence, the recognition module can apply the new relation in order to add the new characteristic to the derived situation and derive the new characteristic's parameter from the other characteristic's parameters (see also Section 4.2.4). Furthermore, also the attention model is modified to add the new characteristic to the focussed situation.

In the example scenario, the CART-algorithm (e.g., see [BFOS84]) is used to generalize assumptions with nominal and numerical parameters. At the beginning, the first situation of a table is used as example to build the model based on the CART algorithm (see Fig. 4.22). Then, the learned model is used to estimate whether the next ten situations fulfill the assumptions. If one prediction is wrong, the model is assumed as not good enough. In the next step an additional situation is used to refine the model and the next ten situations are used to check the model again. This procedure repeats until 200 different situations are used to refine the model.

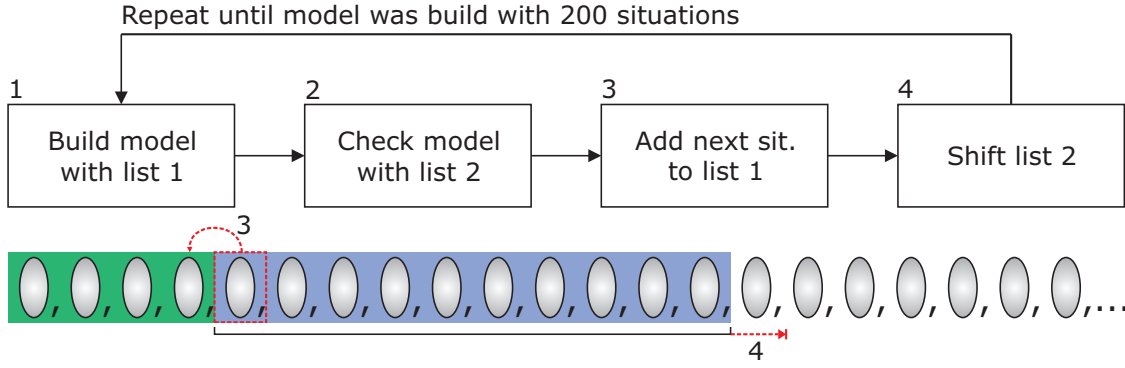


Figure 4.22.: Sequence of different steps in the experiment

In Fig. 4.23, the average success of prediction over the number of considered example situations is illustrated. Furthermore, the learned relation regarding an arbitrary selected table with a certain sequence of situations is visualized as decision tree after 10, 20, and 30 considered situations. The curve shows that a certain minimum number of examples is necessary to build a model with an acceptable accuracy, which depends on the variety of the learned examples. Furthermore, the model is sequentially optimized since the threshold for the characteristic $c_{\text{distanceToFront}}$ is reached more and more precisely. After ten different example situations, the created model is still absolutely wrong since only the characteristic $c_{x\text{-position}}$ is assumed as significant. After 20 example situations, the right characteristics are found and after 30 example situations the threshold for the fourth characteristic is further refined.

Summary

This section presents three different mechanisms to learn the situations' internal structure. If the system processes exclusively characteristics with nominal parameters, a regression analysis can be applied to generalize the assumptions of operators and solve uncertainties in an action space. However, if the system also processes characteristics with numerical parameters, new relations have to be learned. These new relations are used to derive new characteristics with nominal parameters, which may represent the (sometimes complex) connections between the initial situation's characteristics and the operators' functions in a reduced way.

By the proposed approach, a technical system's degree of autonomy can be significantly increased. In contrast to other state-of-the-art approaches, where perception denotes the same as measuring or the application of deterministic interpretation routines, here, the system has the flexibility to adapt its perception automatically to a certain environment.

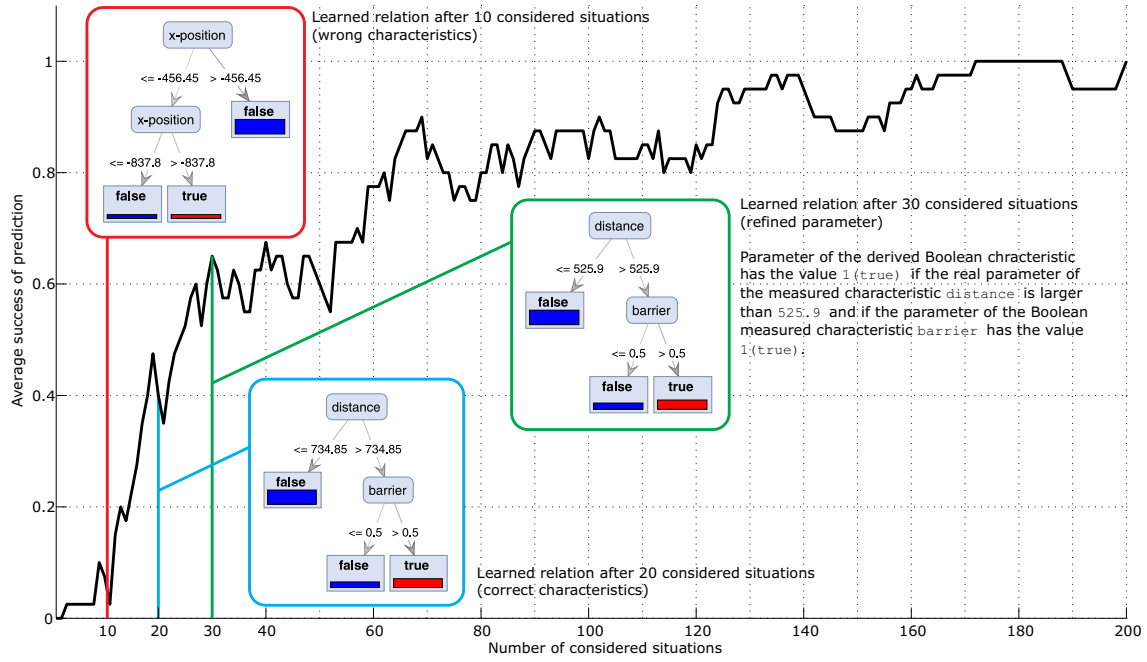


Figure 4.23.: Average success of automatically learned relations in the example scenario

The presented results are used to illustrate the proposed learning mechanisms. However, the approach can also be applied to other kind of systems with different characteristics and operators. How fast the correlating characteristics are detected or how efficiently a general function can be derived, depends primarily on the example situations' structure and sequence. From the application's point of view, here, the performance of the learning algorithms, which can be easily substituted by other data mining methods (like artificial neural networks or support vector machines), is subordinated.

4.4. The cognitive architecture ILCA

By means of the representations and functions provided by the developed framework, central characteristics of human interaction and cognition can be modeled and simulated. Here, the relations among the different functions are intentionally open in order to conserve the flexibility regarding modeling and to avoid that (in the future) new functions have to be forced inside a fixed (and maybe not suitable) structure. Hence, the underlying SOM approach remains as constant exclusively. However, for the realization of a cognitive architecture, it is commonly accepted and necessary that relations among all functions are defined precisely. This is also the case for ILCA, which is an exemplarily implementation of a cognitive architecture based on the developed framework. The acronym ILCA stands for Interactive, Learning, Cognitive Automat and refers to the cognitive architecture's main features including

- interaction with the environment,
- learning capabilities realizing autonomy, and
- simulation of cognitive functions.

Thus, the cognitive architecture is also structured according to the defined intermediate level between real world and technical model. In addition to that, the uniqueness of ILCA results from the combination of several outstanding characteristics as the contained novel kind of knowledge structuring, the related learning mechanisms, and the applicability to arbitrary kinds of technical systems.

In the following, the structure of the ILCA architecture is presented from two different perspectives regarding the representational level on the one side and regarding the cognitive functions on the other side. After that, the processing sequence of the different modules is described by the main cognitive cycle, which consists of a cognitive process, an interaction process for the realization of skill-based behaviors, and two subprocesses for the realization of learning. Here, the cognition process realizes rule-based as well as knowledge-based behaviors and combines the modules for perception, goal selection, goal generation, action space generation, and planning. Besides the four primary processes of the main cognitive cycle, the architecture may also be extended by functions realized as secondary processes. Some examples of those secondary processes are described at the end of this section.

4.4.1. Structure of the representational level

The cognitive architecture's representational level is divided into a short-term memory, a long-term memory, and a working memory (see Fig. 4.24). Hence, the information processing can be related to the currently relevant knowledge and does not have to process one large database. Furthermore, a memory with continuously changing information exists. The different memories contain one or several of the SOM-based representations

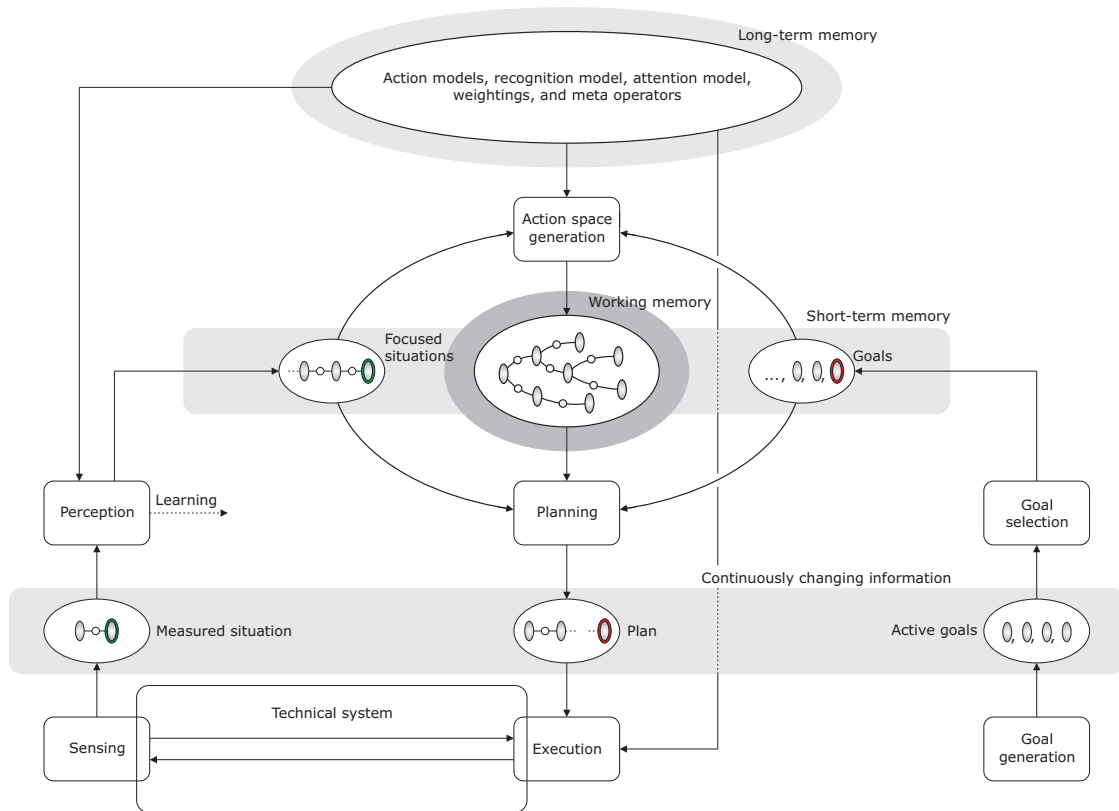


Figure 4.24.: Memory-oriented connection between representation and functions

described in the previous section. Thus, the SOM-based representations can also be considered as submemories.

The relations among the submemories and the cognitive functions are illustrated in Fig. 4.24. Situations and experiences are generated by a sensing module. The situations are further processed by the perception module (see Section 4.2.4) in order to derive further characteristics and to establish the focused situation. This is based on the knowledge stored in the perception model (subdivided into recognition and attention model), which is part of the long-term memory. The focused situation and the executed/observed operator are stored in a list of experiences which is part of the short-term memory (see Section 4.3.4). The second part of the short-term memory is a list of goals. The first element of the list is the current goal and the following elements are previous goals. The working memory is an action space (see Section 3.1.3) which is generated by a module combining information from the short-term and long-term memory. Since the working memory represents the mental counterpart of the real worlds interaction, it is also denoted as ‘mental action space’ (see Def. 4.2). The long-term memory consists of the action models, the perception model, weightings, and meta operators. (see Section 4.2.1) Finally, the working memory and the short-term memory are used to gen-

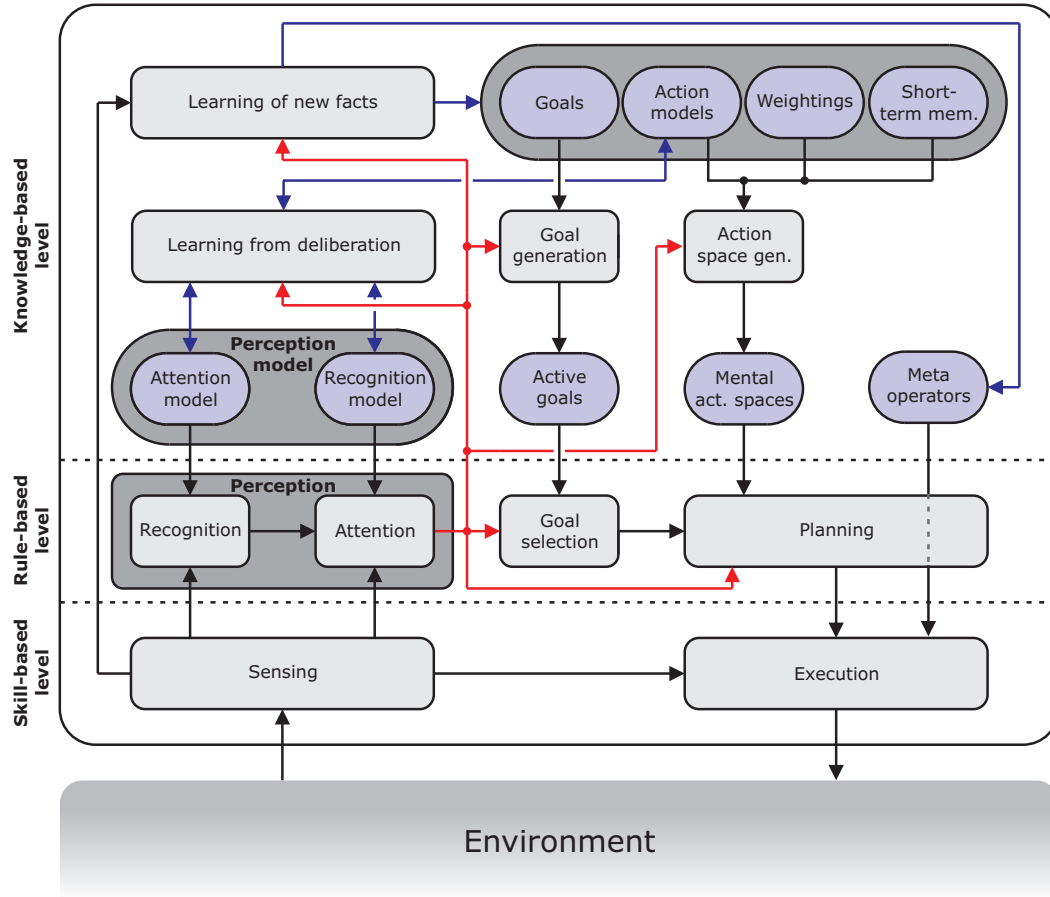


Figure 4.25.: Three-level-structure of the cognitive architecture's modules

erate the current plan to be executed. The plan appears as a list of experiences and is successively executed by the execution module.

Definition 4.2: Mental action space

A mental action space is an action space (see Def. 3.3) which is related to a mental representation of the real world.

4.4.2. Structure of the cognitive functions

The proposed cognitive architecture (see Fig. 4.25) comes with the known three levels for skill-based, rule-based, and knowledge-based decision behavior introduced by RASMUSSEN [Ras83]. Hence, also more simple kinds of agents (simple reflex agent, model-based reflex agent, etc. [RN03a]) can be realized if the highest and highest two levels respectively are neglected. Nevertheless, for the realization of a cognitive system with a high degree of autonomy, the modules on all three levels have to be utilized.

On the skill-based level, a situation is established from sensor measurements which can be partially prefiltered. Furthermore, operators (with additional parameters or not) are executed as actions in the real world. Hence, the modules for sensing and execution are the interfaces between the architecture and a connected technical system's sensors and actuators. Here, the arrow between the sensing and the execution module indicates the performance of sensomotoric actions and fast reflexes, which are implemented directly at the technical system (see Section 5).

On the rule-based level, modules for perception, goal selection, and planning are implemented. Here, the perception module is further divided into two modules for attention and recognition. In all four modules, rules are used to interpret the measured situation from the sensing module (recognition), to focus on those characteristics which are relevant in the current situation and/or for the next operator (attention), to reorient the behavior (goal selection), and to generate a sequence of actions to be performed (planning module). The applied rule-based knowledge can be previously defined by the system designer or learned from interaction with the real world.

The knowledge-based level becomes important if the internally represented mental world does not correspond to the real world, e.g., if a plan to a desired situation can not be generated or if ambiguous knowledge exists. Furthermore, the system may continuously learn new facts from interaction with the environment. The knowledge of the architecture comprehends several models which are related to different memories (see above). In this context, the models can represent general knowledge or rule-based knowledge (see Fig. 4.25). All of these models are based on the SOM approach as common methodical background and they together build the mental model of the system. Except the mental action space, all models can be directly modified by two learning modules. However, also the mental action space may change since it is generated from the general knowledge changed by the learning modules.

4.4.3. Main cognitive cycle

In the previous paragraphs, the structure of the architecture's representational level and cognitive functions is described. However, for the realization of a cognitive system, also a certain procession order has to be defined, which is denoted in the following as the main cognitive cycle. The cognitive architecture is implemented within one computer program communicating with other software (e.g., controller and/or filter programs of a technical system) over a network connection. This program realizes the cognitive cycle by using the modules of the developed framework and is divided into the four primary processes

1. cognition,
2. interaction,
3. learning of new facts, and
4. learning from deliberation.

The primary process cognition contains the modules for perception, goal selection, goal generation, action space generation, and planning. Here, the focused situation is established and it is checked whether the current goal is still valid. If a new goal is selected or if an unexpected situation occurred, a new action space and a new plan is generated. After a planned action was executed by the interaction process, the deliberation process starts again in order to process the sensed measurements by the perception module. Both learning processes may be started optionally within the cognition process. Learning of new facts starts directly after the focused situation was established and is used to integrate new facts as the current situation and experience into the system's mental model. Learning from deliberation may start after the learning of new facts or after the planning module. In both cases, the systems knowledge may be refined by generalizing the assumptions of known operators or by the resolution of conflicts (uncertainty about the real world's dynamic).

Cognition

The primary process cognition utilizes the modules of the developed framework in order to realize rule-based and knowledge-based behaviors. As long as the available action spaces are applicable to generate plans which can be moreover executed successfully, the behavior may remain completely rule-based. Otherwise, the system has to switch to the knowledge-based level. Here, new action spaces (with different degrees of abstraction) can be generated and learning mechanisms are applied.

The cognition process starts after the interaction process, which provides the measured situation. At first, the measured situation is processed by the perception module to establish the current focused situation and an experience representing the considered effects of the last action. These information are integrated in the system's knowledge by the subprocess learning of new facts, but they do not have any influence to the action planning if the old action space remains. After that, the goal selection module checks whether the current goal is still valid or another goal (e.g., with a higher priority) has to be aimed. If a plan exists from a previous cognition process (including the current situation) and if the related goal is still valid, the next operator can be simply selected and executed. Otherwise, a new plan has to be generated based on the current situation, goal, and mental action space. Here, also the short-term memory and meta operators may be considered.

If planning fails due to the fact that the mental action space does not contain a path from the current situation to the desired situation (which happens automatically if a subgoal is reached), the system has to switch to the knowledge-based level to generate a new mental action space based on the current situation. Whether only the action space with the lowest degree of abstraction is updated or if also action spaces on higher degrees of abstraction are affected depends on the current goal. Nevertheless, in all cases, also the new facts learned from interaction are taken into account. Hence, the obligatory generation of a new mental action space before planning may be helpful in any case (also if the old mental action space contains a path to the current goal).

If the planning still fails after the mental action space was regenerated, the operator's assumption represented in the action models can be generalized systematically by the subprocess learning from deliberation. If planning still fails after learning from deliberation, which may happen if a completely new environment is explored, the next action can also be chosen according to a certain (maybe random-based) selection pattern. However, due to the generalization and the learning of new facts, the system may represent ambiguously knowledge. In this case, the existing conflicts can also be solved by the subprocess learning from deliberation. Both learning mechanisms, the generalization of assumptions and the solution of conflicts, may be executed optionally within the cognition process and lead to the successively refinement of the system's mental model.

Interaction

If a plan exists, the contained operators can be executed successively by the primary process of interaction. Here, the architecture sends the name of the operators (with additional parameters if necessary) to the technical system. Then, the architecture waits until the technical system received the command and until the corresponding action was executed respectively. After the execution, the architecture receives a message containing the information in which way the operator was finished. Besides the information that the operator was executed successfully, the message can also contain that the operator's assumptions were not fulfilled or that the operator had to be aborted.

From the application of the operator, the required time for execution and new parameters of the measured characteristics result. The measured characteristics are processed in the subsequent cognitive process by the perception module to establish the current focused situation. Furthermore, the current focused situation is used to establish the current experience, which is provided with the required time for execution to the first learning process (learning of new facts).

Due to the underlying SOM approach, the presented process of interaction only executes operators sequentially. However, this does not imply that a technical system to be controlled is only able to control its actuators in a sequential order. Since the proposed system is able to focus different characteristics depending on the current situation

and the operator to be performed, the focused situation may be constant although one or several actuators are controlled and certain characteristics are changed respectively. Hence, also complex behaviors can be executed if several operators are executed sequentially in order to start or modify several control routines. In this regard, also the parallel execution of several interaction processes or several main cycles (as interacting agents) would be possible extensions.

Learning

After an action was performed within the real world, new facts from the sensed information have to be integrated in the system's mental model and they are used to derive more general information or to solve uncertainties. Hence, learning is realized within the proposed cognitive architecture by two different processes including

1. learning of new facts, and
2. learning from deliberation.

The available facts consist of the current experience, the system's plan, and the time required for execution (as an example for a certain weighting). Here, the current experience was provided by the perception module and consists of the previous situation, the name of the applied operator, and the current situation. The learning from deliberation may be executed after the system's mental model was updated by new facts or alternatively after the system was not able to generate a mental action space or plan respectively. In this case, planning is executed a second time within one cognitive process based on the modified knowledge.

In order to integrate new facts in the system's mental model, several steps are executed. At first, the current experience is added to the short-term memory. If the contained final situation was moreover not expected (not contained in the plan), the current experience is also integrated in the action model. Here, uncertainties can occur, which lead to the generation or update of corresponding classes in order to solve the conflict afterwards. Furthermore, if the current situation corresponds to the current goal and if the executed action was planned by the system, the corresponding plan is stored as a meta operator. Finally, the required time for executing the operator is integrated within a list of weightings.

If a new experience was added to the action model, the system can try to generalize the assumptions of the related operator. Thus, the system is enabled to apply this operator also to situations which were exclusively generated mentally (not measured before). Furthermore, if an existing conflict was updated, the system can try to solve the conflict by the integration of new (previously not considered) characteristics in the focused situation in order to reduce the uncertainty about the real world's dynamic.

4.4.4. Secondary processes

In addition to the main cognitive cycle with the presented primary processes, some of the contained cognitive functions may also be realized as secondary processes. These secondary processes can be considered as independent threads interacting with each other and the main cognitive cycle. Hence, it is possible to consider a certain problem from different perspectives, e.g., with different degrees of abstraction or based on different models. Furthermore, cognitive functions as perception could also run in parallel to the interaction process in order to stop the execution if necessary. In the following, four different processes are presented exemplarily. The first three processes can observe internal and external states of the system and may interrupt the main cognitive cycle to handle the corresponding exception or problem. The fourth process performs parallel computing and generates additional knowledge or solutions.

- The first kind of secondary process is an alternative implementation of the cognitive function of perception. Thus, the system can permanently observe the current situation independently of other functions of the main cognitive cycle. For example, if the current situation is identified as dangerous (e.g., by comparing the perceived situation with a database of dangerous or undesired situations), the main cognitive cycle can be interrupted in order to plan and execute an alternative sequence of operators.
- Also the generation and selection of goals can be realized as secondary process. Since the current situation may change during the interaction process, also a new goal can be activated and selected if its priority is higher than the current goal. If this is the case, the main cycle can be interrupted in order to generate a new plan leading to the new goal.
- A further kind of secondary process could use the current situation, goal, and mental action space in order to search for alternative paths which would lead to the goal more quickly or secure. If such an alternative solution is detected, it might be useful to stop the currently executed operator and generate a new plan.
- Finally, secondary processes may also be used to generate new hypothetic knowledge from existing knowledge, which is the case for computationally intensive functions as generalization or conflict resolution (depending on the complexity of the system). If these calculations are only used to optimize the system's behavior and are not required to reach the current goal in a better way, they could also be executed by secondary processes running in parallel to the main cognitive cycle.

All of the proposed secondary processes can be realized by the functions and representations of the developed framework. Neither the number nor the type of secondary processes is fixed. Hence, the concept offers a comfortable way to integrate additional functionalities to the cognitive architecture.

5. Realization of Cognitive Technical Systems

A large number of cognitive architectures are applied to realize cognitive control for technical systems (see Section 2). In this regard, the kinds of implementation vary such as the kind of applied architectures. Some architectures have interfaces with sensors and actuators and some architectures are parts of other software systems communicating with technical devices. In some approaches, architectures are applied, which are originally designed for human performance modeling and in other approaches, functions are contained, which are exclusively designed for a certain technical system. However, there is no system available that can be simply connected to arbitrary technical systems to perform cognitive control and to learn a mental model automatically. In order to address this issue, the ILCA architecture has a general interface, which can be used to communicate with a large number of different kinds of technical systems (see Fig. 5.1).

This chapter describes how to connect the proposed cognitive architecture with technical systems in general. Therefore, it is detailed in which way the technical system has to be specified and how the communication between architecture and technical system is solved. As an example, the cognitive control of a mobile robot interacting with a dynamical environment is chosen. After a brief introduction to the robot's hard- and software, the example scenario and the defined characteristics and operators are given. The example scenario is also referred by the following chapter to demonstrate the developed planning and learning functions.

5.1. Specifications of technical systems to be controlled

Cognitive systems have self-awareness available, which allows them to reflect on their own behavior and the behavior of their environment. In this respect, the related conscious parts of the cognitive functions of planning or learning do not have to be free of errors and do not have to be performed in real-time. In contrast to that, some sensomotor behaviors as reflexes or trained skills are performed very fast and in parallel. The behaviors are unconscious and the related underlying knowledge is represented implicitly. Not every sensed fact is considered in detail and only abstract versions of these behaviors are processed consciously and represented in an explicit manner.

The proposed cognitive architecture represents and processes explicit and implicit knowledge as well. However, if a certain behavior has to be performed in real-time, a direct connection of sensors and actuators is necessary. This again results in additional implicit knowledge to be implemented. Hence, the communication among sensors, actuators and the cognitive architecture has to be designed appropriately.

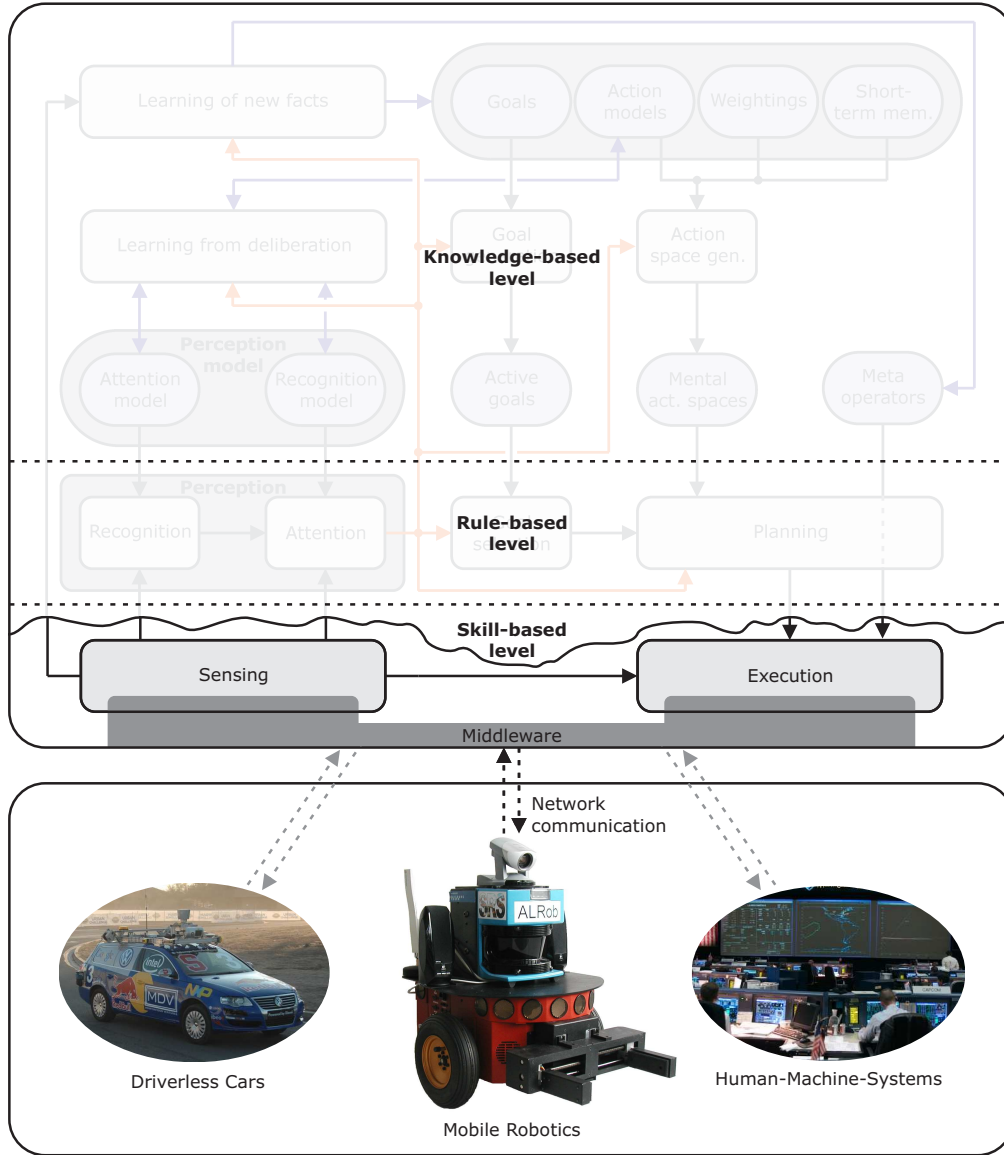


Figure 5.1.: Cognitive architecture for arbitrary kinds of technical systems

Due to the fact that the proposed cognitive architecture should be able to control different kinds of technical systems, a standardized interface to sensors and actuators is necessary. Therefore, the proposed cognitive architecture makes use of a middleware (see [HS10]) allowing network communication among different software almost independently from programming language or operating system. Via the middleware the cognitive architecture receives the current situation as a list of characteristics and sends the operators or operator sequences as commands with additional parameters. The approach assumes one or several control programs which communicate with one or several sensors and/or actuators in real-time and communicate via the middleware with the cognitive architecture (see Fig. 5.2). Consequently, the technical system to be controlled

needs an appropriate communication port and a control unit supporting programs that contain an interface to the used middleware (as provided by a usual personal-computer). The control programs could work as filters and/or controllers, which are used or started by the cognitive architecture.

A filter program (or subroutine) communicates with one or several sensors providing continuous or discrete signals which correspond to certain effects of the physical world. The received signals are used to calculate the parameters of measured characteristics. Here, signals can also be fused, interpreted, or used to extract new features [GGNZ06]. The measured characteristics are locally stored and they can be provided to the cognitive architecture or to a controller subroutine.

A controller program (or subroutine) is started by the cognitive architecture and realizes an action of the technical systems. The action is represented in the cognitive architecture as basic operator. If a controller program has one or several parameters influencing the corresponding action of the system, several basic operators are represented. In order to perform an action, a controller program uses sensor signals and/or the parameters of measured characteristics (provided by a filter program or subroutine) to control the actuators. Due to the fact that the sensor signals are filtered, the control algorithms should be designed in a way, so that they also change one or several of the measured characteristics. Hence, it can be guaranteed that initiated changes in the real world are also observed by the cognitive architecture in order to learn the function of the mapped action. Due to safety reasons, not every control program can be started in

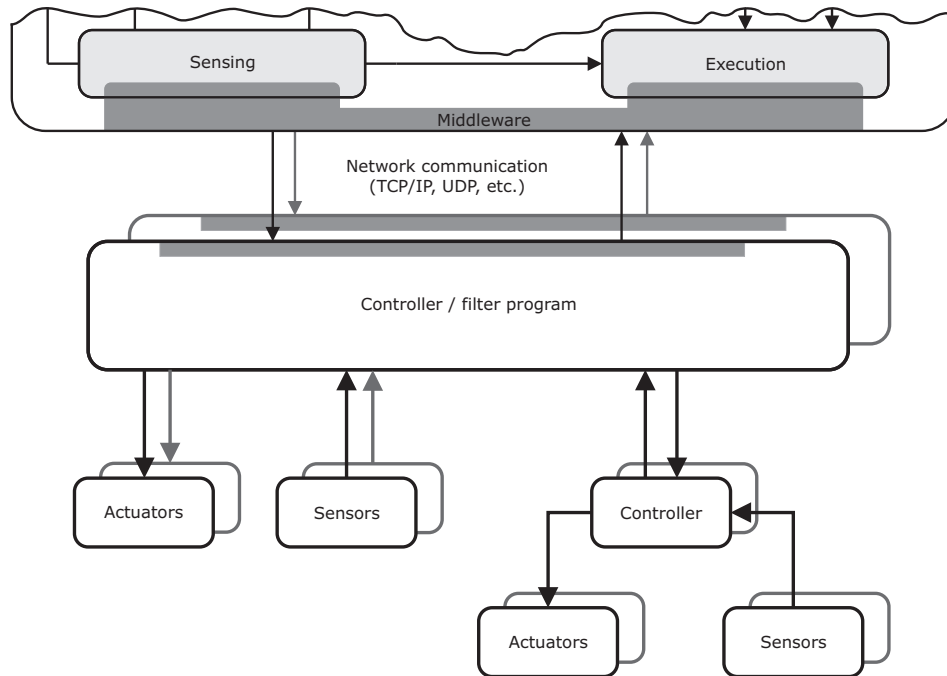


Figure 5.2.: Architecture communicates with technical system(s) over middleware

every situation. Hence, the parameters of some relevant characteristics (that have to be defined by the system designer) can be checked at the beginning of the related control program to prevent that the control algorithm starts and causes a dangerous situation. In order to monitor a controller program by the cognitive architecture, it should also be possible to check whether a controller program is still running or already finished with or without errors. Finally, the controller programs can also be used to realize reflexes in real time (e.g., to avoid dangerous situations) implemented by hard-coded rules.

In the described approach, the control programs are defined in a ‘hard-wired’ way. They can not be changed during runtime such as the electrical and mechanical parts of a technical system. Nevertheless, the control programs could also be defined in a more variable way in order to realize skill-based learning by the fusion of trajectories or methods from adaptive control [AW94]. However, due to the fact that the implementation can be very system specific, this thesis focuses on the learning mechanisms which can be realized in the cognitive architecture.

5.2. Realization of a cognitive mobile robot

As an application example, a cognitive control system consisting of the ILCA architecture and a Pioneer 3 DX (see [Mob10]), which is a commercial mobile robot with several sensors and actuators, is presented. In the following subsections, the hardware and software components of the whole system realizing a cognitive mobile robot are described. Afterwards, the operators and measured characteristics of an example scenario are defined. The same scenario is also used as the basis for practical experiments, which are presented in the following chapter to demonstrate the developed cognitive functions.

5.2.1. Hardware

The whole system consists of the Pioneer 3 DX (see Fig. 5.3) containing an onboard PC (see [Ver10]) and an additional laptop where the ILCA architecture is installed. This structure is simply motivated by some advantages regarding the evaluation of experimental data and a clear separation of technical system and cognitive architecture as well. On the onboard PC, a control program with several filter and controller routines runs, which communicate (partially) over a micro controller with the robot’s sensors and actuators. The technical communication between control program and cognitive architecture is realized by a WiFi connection. Therefore, the robot is equipped with a network card and an antenna, which restricts the robot’s operation area to a certain distance to the laptop’s location. However, this operation area is large enough for the chosen example scenario. Furthermore, the operation area can be enhanced arbitrarily if the cognitive architecture it is running on the onboard PC.

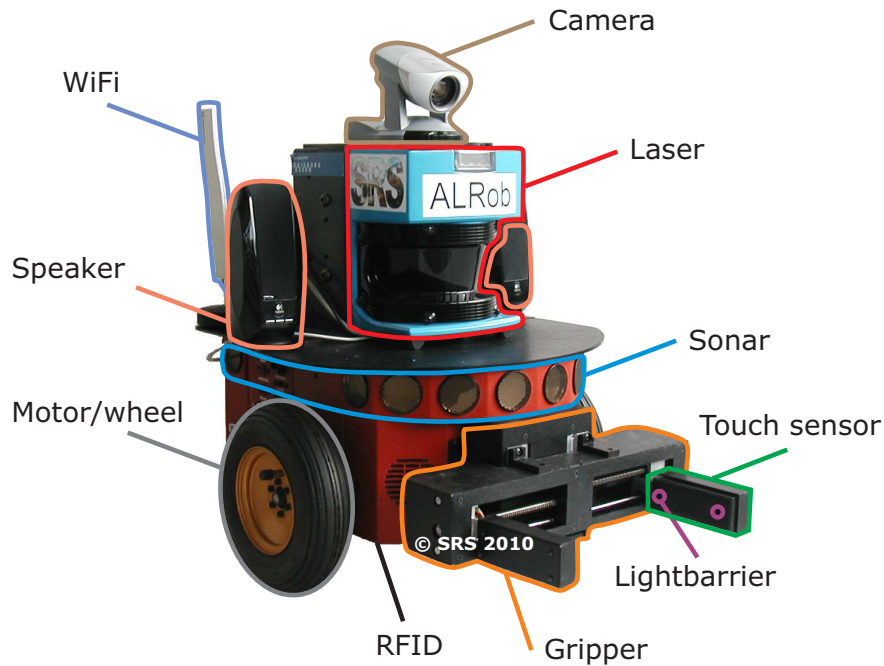


Figure 5.3.: Sensors and actuators of the mobile robot

The mobile robot contains three lead-acid secondary batteries, each with a capacity of 7,200 mAh. Depending on the scenario to be realized and the batteries state-of-health, the robot can perform approximately two hours independently from an external energy supply. However, a complete charging takes much longer, especially if the onboard PC is running. Hence, the robot has to be charged in regular periods to realize longer operation times (which automatically reduces the batteries' lifetime extremely).

The system is mechanically designed for the navigation in office environments and it is also equipped with several sensors and actuators to perform related tasks. In order to move transversally and radially in a building, the robot contains two large wheels mounted laterally and a small freemoving wheel at the back side. The large wheels can be controlled independently from each other by two motors, whose rotation is measured by odometry sensors, which are applied to estimate the driven way roughly. Due to several influences as slipping wheels or measurement errors, the internal representation of the system's position in the environment has to be adjusted continuously. Possible solutions for this are probabilistic approaches estimating the robots position from a geographical map and distance measurements. In order to measure the distance to other objects, the robot is equipped with 16 ultra sonic sensors and a laser range finder. The ultra sonic sensors are mounted around the robot and they can be used to measure the rough distance to obstacles on different heights. Hence, also chairs and tables can be detected. In contrast to the ultra sonic sensors, the laser range finder can detect the distance to objects very precisely, but only on a certain height and only at the robot's

front side within a range of 180° . Besides the distance, also the color and the shape of objects can be measured by a controllable video camera (see [Mob04]). Finally, objects can be gripped and lifted by a gripper (see [Mob07]) mounted at the front of the robot, which contains two light barriers and a touch sensor. Hence, the robot can carry objects if they have a suitable size and weight.

Besides the sensors and actuators provided with the Pioneer 3 DX system originally, the robot is equipped with additional components as a RFID-reader, an I/O-card and two speakers. The RFID-reader generates an electromagnetic field and detects whether a transponder-tag is within this field. The used transponder-tags are smart labels which can be read or written. Hence, the RFID-reader can be used to detect fixed landmarks in order to estimate the location of the robot. Alternatively, also a probabilistic approach (included in ARNL [Mob09], an API provided by the robot's manufacturer) could be applied. However, due to the fact that this approach makes use of static geographical maps which do not represent a dynamical changing environment, the localization through landmarks is chosen. The RFID-reader is connected with the onboard PC by a separate USB-port. Depending on the orientation of the RFID-reader, transponder-tags within a range of up to one meter can be detected. However, since the RFID-reader is mounted on a metallic body containing several devices with electromagnetic radiation (disturbing the electromagnetic field of the RFID-reader) and the robot is moving during measuring, the measuring range is reduced extremely. Hence, the RFID-reader is located beneath the robot in order to detect transponder-tags on the corridor's ground with a high probability.

To realize simple interaction with humans, an additional I/O-card and two speakers are added to the system. The speakers can be used to generate acoustic signals, e.g., to warn passersby. Furthermore, simple statements or questions can be addressed to humans. In order to answer the robot's questions or to give certain information (e.g., to define a goal), two buttons are mounted on the robots back side and connected to the I/O-card. The I/O-card is connected again with the onboard PC's USB-port and contains eight digital inputs, eight digital outputs, and eight analog inputs. Hence, the connection of further sensors and/or actuators with the robot is relatively simple.

5.2.2. Software

The connection between cognitive architecture and technical system is realized by the used middleware providing a network communication of a server and several clients. Here, the cognitive architecture acts as the server and runs on the external laptop. Although the cognitive architecture (receiving measured characteristics and sensing commands to execute operators) could communicate with several control programs acting as clients, only one control program is necessary in the presented example system since all required software components are available for the same programming language. The control program contains several controller and filter routines and is running on the

robot's onboard PC. In order to communicate with the sensors and actuators of the robot system, the filter and controller routines use several API's (Application Programming Interfaces). The most important API is ARIA, which is provided with the Pioneer 3 DX system. The other API's are used to communicate with the I/O-card and the RFID-reader.

The classes provided by ARIA can be used within a C/C++ program to request and control the robot's motors, gripper, ultrasonic sensors, laser range finder, video camera, and power supply system. Some of the classes' methods initiate routines in the robot's microcontroller that again communicates with the technical devices on the lowest level. Hence, the actuators can be operated in real time. In order to detect colors, the software ACTS (see [Mob08]) runs in parallel to the control program and is also accessible through methods from ARIA. The software receives the video signal from the robot's camera and provides information about color blobs. Here, a blob is a certain area in the video image, which exclusively contains pixels with the same color. In the same way as the software ACTS, also other software tools may be added to the system.

The I/O-card and the RFID-reader are accessed by separate API's and the generation of sounds is initiated by commands from a standard C/C++-library. Nevertheless, the complete code can be combined in one control program and it is used by several controller and filter routines. If one of the I/O-card's digital inputs is closed only for a short time, a corresponding internal variable representing the parameter of a measured characteristic is set to true. Hence, this function can also be used to stop the execution of a certain operator or to set a previously defined goal at the robot system directly. Analogically, the API of the RFID-reader is used to query the ID of a transponder-tag if one is detected within the generated electromagnetic field. Additionally, detected transponder-tags could also be written in order to use the RFID-reader as actuator.

5.2.3. Example scenario: office navigation and service

As an example for a complex and dynamical real world scenario, the navigation and performance of service tasks in an office environment is chosen. The mobile robot has to drive among different rooms which are connected by a circular corridor (see Fig. 5.4). Furthermore, the robot has to pick up objects from certain offices and has to place them in another office or the kitchen. The defined standard task is the collection of empty bottles from the offices. If an empty bottle is found, it has to be placed at a certain position in the kitchen. Additionally, the robot has to rest in regular time intervals in order to charge the batteries in one of the offices. Therefore (and for other tasks), also the communication with humans is necessary since the system does not have an actuator available, which can be used to connect itself with the power supply unit automatically.

The complexity of the scenario arises from the fact that the chosen corridor has no clear structure as a laboratory environment. Due to the fact that also humans interact



Figure 5.4.: Mobile robot navigating through the office environment

independently from the robot within the same environment, the interaction of the robot has a dynamic, uncertain, and open character. Human passengers can cross the robot's way and influence whether office doors are opened or closed. Furthermore, humans usually do not have simple behavior patterns, their intention is hard to measure, and they can change the environment in an (for the system) unexpected manner. Hence, the generation and usage of static navigation maps is not suitable for localization and path planning.

In the following, the system's measured characteristics and defined operators are described. These are necessary to perform successfully in the advised scenario and to learn an appropriate mental model containing all relevant facts and relations in order to enhance the system's behavior.

Measured characteristics

The robot's sensors measure different physical effects which are further processed by the control program's filter routines providing the parameters of the resulting measured characteristics to the cognitive architecture. In the following, the measured characteristics describing the scenes of the chosen example scenario as unambiguous and clear as possible are explained in detail. Furthermore, Table 5.1 summarizes all measured characteristics, their range of parameters, and related sensors.

- With the video camera the four colors red, yellow, green, and blue can be detected (e.g., $c_{\text{redObjDet}}$). Furthermore, their vertical orientation to each other (e.g., $c_{\text{posOfYellowObj}}$) can also be taken into account.

- With the laser range finder, the shortest distance to an obstacle in a certain direction can be measured. Here, the distances to objects in front and on the sides of the robot are used (e.g., $c_{\text{obstacleInFront}}$). Furthermore, an additional characteristic $c_{\text{accessibleAreaInFront}}$ is used to describe whether the area in front of the robot is accessible in general (e.g., open door in front) or not (e.g., closed door in front or wall in front).
- By the odometry sensors, the x-position (c_{xPos}) and y-position (c_{yPos}) of the robot regarding the initial coordinates are estimated. Furthermore, the robot's orientation ($c_{\text{orientation}}$) within the environment is derived. All three characteristics have numerical parameters, which are strongly influenced by integrating measurement errors.
- If a RFID-tag is detected, c_{tagDet} is set to true and $c_{\text{idOfDetTag}}$ stores the related ID of the tag.
- The gripper has two light barriers (one at the front and one at the rear part) which can be free or blocked (e.g., $c_{\text{firstBarrierClosed}} = \text{true}$). Additionally, with $c_{\text{objGripped}}$, it is described whether the gripper is opened or closed.
- In $c_{\text{timeOfDay}}$, the current time of day (morning, noon, and afternoon) is contained. Hence, the behaviors of different humans depending on the time of day, e.g., not in the office before 6.00 am and after 8.00 pm, can be considered.
- The batteries' state-of-charge is represented by the characteristic $c_{\text{batteryStatus}}$. Depending on the measured cell voltage, the related parameter is set to empty, middle, or full. Furthermore, the characteristic $c_{\text{isCharging}}$ changes based on sudden changes of the cell voltage or related information from a human which connected/disconnected the robot with the power supply.
- The characteristic $c_{\text{lastOperator}}$ stores the previous action of the robot. Here, the parameter of $c_{\text{lastOperator}}$ changes after $o_{\text{createSound}}(\text{pleaseConnect})$ to 1, after $o_{\text{createSound}}(\text{pleaseDisconnect})$ to 2, after $o_{\text{createSound}}(\text{openDoor})$ to 3, after o_{charging} to 4, and after all other operators to 0. Hence, it is also possible to represent the dependency of a certain operator to its predecessor. If the robot drives into a room, the characteristic $c_{\text{insideRoom}}$ is set to true and the corresponding room number (detectable by RFID-tags before the rooms) is stored in $c_{\text{noOfCurRoom}}$. Furthermore, the characteristic $c_{\text{relOrientation}}$ changes if the robot turns. The characteristic describes whether the robot is oriented to the center of the office environment, to the corridor (in clockwise (cw) and anticlockwise (acw) direction respectively), or to the windows (see Fig. 5.4).
- Finally, the answers and commands of humans are represented by two characteristics describing whether a related button was pressed or not (e.g., $c_{\text{firstButtonPressed}} = \text{yes}$).

Source	Characteristic	Range of parameters
Camera	$C_{redObjDet}$	{true, false}
	$C_{yellowObjDet}$	{true, false}
	$C_{greenObjDet}$	{true, false}
	$C_{blueObjDet}$	{true, false}
	$C_{posOfRedObj}$	{nil, top, middle, down}
	$C_{posOfYellowObj}$	{nil, top, middle, down}
	$C_{posOfGreenObj}$	{nil, top, middle, down}
	$C_{posOfBlueObj}$	{nil, top, middle, down}
Laser	$C_{obstacleInFront}$	{close, between, far}
	$C_{obstacleLeft}$	{close, between, far}
	$C_{obstacleRight}$	{close, between, far}
	$C_{accessibleAreaInFront}$	{true, false}
Odometry	C_{xPos}	$\{p \in \mathbb{R}\}$
	C_{yPos}	$\{p \in \mathbb{R}\}$
	$C_{orientation}$	$\{p \in \mathbb{R} \mid 0 \leq p < 360\}$
RFID	$C_{idOfDetTag}$	$\{p \in \mathbb{N} \mid 0 \leq p\}$
	C_{tagDet}	{true, false}
Gripper	$C_{firstBarrierClosed}$	{true, false}
	$C_{secondBarrierClosed}$	{true, false}
	$C_{objGripped}$	{true, false}
System	$C_{timeOfDay}$	{morning, noon, afternoon}
	$C_{isCharging}$	{true, false}
	$C_{batteryStatus}$	{empty, middle, full}
Operators	$C_{lastOperator}$	$\{p \in \mathbb{N} \mid 0 \leq p\}$
	$C_{noOfCurRoom}$	$\{p \in \mathbb{N} \mid 0 \leq p\}$
	$C_{insideRoom}$	{true, false}
	$C_{relOrientation}$	{clockwise, center, anticlockwise, windows}
I/O-card	$C_{firstButtonPressed}$	{true, false}
	$C_{secondButtonPressed}$	{true, false}

Table 5.1.: Characteristics defined for the scenario

Operators

The operators represented by the cognitive architecture model the actions of the controlled technical system. In the case of the chosen scenario, the robot has to perform pick and place tasks in a specific dynamical office environment. Hence, the system has to

- drive on the corridor from room to room,
- change its orientation by turning 90° or 180°,
- drive inside and outside rooms,
- pick up and drop objects as bottles,
- generate different kinds of sounds, and
- wait for certain changes in the environment.

The assumptions of the corresponding operators can be divided into assumptions defined by the system designer to protect the system and assumptions resulting from a certain environment. The first type of assumptions is defined in the controller routines in order to avoid dangerous situations. In contrast to that the second type of assumptions can not be defined in advance since they result from the system's interaction with the environment. However, the cognitive architecture does not distinguish between different types of assumptions if they are derived from the system's experiences.

In the following, the functions and assumptions of all operators are described in detail. Furthermore, Tab. 5.2 gives a brief summary by relating each operator to their user-defined assumptions and to the characteristics whose parameters are changed by the operator.

- **o_{driveToRoom}(direction)** With this operator, the robot can drive along an arbitrary formed corridor by following the wall. If an open or closed door is detected (here, realized by RFID-tags on the floor), the robot stops. As assumptions for this operator, the robot has to stand near an RFID-tag (e.g., $c_{noOfDetTag}=3$) and besides a wall (e.g., $c_{relOrientation}=clockwise$). Furthermore, no obstacle, excluding a gripped object, may stand in front of the robot ($c_{obstacleInFront}=false$).
- **o_{turn}(direction), o_{changeDirection}()** Through these operators, the robot is turned about a certain angle. Depending on the parameter 'direction', the first operator turns the robot +90° and -90° respectively. As assumption, a RFID-tag has to be detected (e.g., $c_{noOfDetTag}=2$). The second operator turns the robot through 180°. Here, the robot has to be inside a room (e.g., $c_{insideRoom}=true$).
- **o_{createSound}(sound)** This operator is used to create different (depending on the parameter 'sound') predefined sounds, like a signal-horn or instructions/questions to humans. It has no assumptions and it can be performed in every situation.

- **OwaitForAnswer(question), OwaitForFreeSpace()** These operators are used to wait for actions from humans. In both cases, the system waits for the corresponding change of the situation (e.g., $c_{firstButtonPressed}=true$ or $c_{accessibleAreaInFront}=true$ respectively) without any own actions. If the situation is not changed adequately, the operators finish after a certain time interval. As assumptions, the operator $O_{createSound}(sound)$ has to be performed beforehand (e.g., $c_{lastOperator}=1$). The second operator requires additionally that the space before the robot is not free ($c_{accessibleAreaInFront}=false$).
- **Ocharging()** Similar to the operators described above, this operator waits until the parameter of the characteristic $c_{batteryStatus}$ changes from empty to full. Alternatively, the operator can be configured in the way that it stops after a certain time. The operator can only be used after the operators $O_{createSound}(pleaseConnect)$ and $O_{waitForAnswer}(connection)$ were applied.
- **OdriveInsideRoom(), OdriveOutsideRoom()** Through these operators, the robot drives inside and outside rooms, like offices or the kitchen. To use the first operator, the robot has to stand on the corridor before an open room (e.g., $c_{noOfDetTag}=4$ and $c_{accessibleAreaInFront}=true$) and has to be orientated to this room ($c_{relOrientation}=windows$). In order to use the second operator, the robot has to be inside a room (e.g., $c_{noOfCurRoom}=2$) and also has to be orientated to an open door (e.g., $c_{relOrientation}=center$ and $c_{accessibleAreaInFront}=true$).
- **OdirectToDoor()** The operator can be used to direct the robot straightly to an open door. If the robot turns to an open door by applying the operator O_{turn} , it could happen that the robot is not straightly directed to the door. In this case, the open door is recognized correctly ($c_{accessibleAreaInFront}=true$), but one of the door-jambs is detected as obstacle ($c_{obstacleInFront}=true$). Hence, the assumptions to pass the door are not fulfilled, which can be changed by the mentioned operator $O_{directToDoor}$. The operator can be applied if an open door is detected in front of the robot ($c_{accessibleAreaInFront}=true$).
- **OgripObject(color), OdropObject()** These operators are used to pick and place colored objects, like boxes or bottles. Depending on the parameter 'color', the first operator orientates the robot to the corresponding colored object, drives to the object, and lifts it up with the gripper. For the execution of this operator, a colored object has to be detected (e.g., $c_{greenObjDet}=true$). To place a gripped object, the second operator opens the gripper and drives a few centimeters backwards. As assumption for this operator, an object has to be gripped ($c_{objGripped}=true$).

Operator(s)	Assumptions			Relev. changes
OdriveToRoom(clockwise)	CobstacleInFront	=	far	CidOfDetTag
	CrelOrientation	=	cw	
	CinsideRoom	=	false	
OdriveToRoom(anticlockwise)	CobstacleInFront	=	far	CidOfDetTag
	CrelOrientation	=	acw	
	CinsideRoom	=	false	
Oturn(left)	CinsideRoom	=	false	CrelOrientation
Oturn(right)	ClastOperator	=	0	CobstacleInFront
				CaccessibleAreaInFront
OchangeDirection()	CrelOrientation	=	windows	CrelOrientation
	CisCharging	=	false	CobstacleInFront
	ClastOperator	=	0	CgreenObjDet
	CinsideRoom	=	true	CaccessibleAreaInFront
OdirectToDoor()	CaccessibleAreaInFront	=	true	CobstacleInFront
OcreateSound(openDoor)	CobstacleInFront	=	middle	CobstacleInFront
	CrelOrientation	=	windows	ClastOperator
	ClastOperator	=	0	CaccessibleAreaInFront
	CinsideRoom	=	false	
OwaitForFreeSpace()	CinsideRoom	=	false	CobstacleInFront
	CrelOrientation	=	windows	ClastOperator
	ClastOperator	=	3	CaccessibleAreaInFront
OdriveInsideRoom()	CobstacleInFront	=	far	CobstacleInFront
	CrelOrientation	=	windows	CinsideRoom
	CinsideRoom	=	false	CnoOfCurRoom
	ClastOperator	=	0	CidOfDetTag
	CaccessibleAreaInFront	=	true	CgreenObjDet
				CaccessibleAreaInFront
OdriveOutsideRoom()	CobstacleInFront	=	far	CobstacleInFront
	CrelOrientation	=	center	CnoOfCurRoom
	CinsideRoom	=	true	CidOfDetTag
	CaccessibleAreaInFront	=	true	CaccessibleAreaInFront
OgripObject(green)	CgreenObjDet	=	true	CgreenObjDet
	CinsideRoom	=	true	CobjGripped
	CobjGripped	=	false	
	ClastOperator	=	0	
	CisCharging	=	false	
OdropObject()	CobstacleInFront	=	close	CobjGripped
	CinsideRoom	=	true	CgreenObjDet
	CobjGripped	=	true	
	CgreenObjDet	=	false	

continued on the next page

Operator(s)	Assumptions		Relev. changes
OcreateSound(pleaseConnect)	CnoOfCurRoom	= 1	ClastOperator CisCharging
	CrelOrientation	= windows	
	CisCharging	= false	
	ClastOperator	= 0	
	CobjGripped	= false	
OwaitForAnswer(connection)	ClastOperator	= 1	ClastOperator CisCharging
Ocharging()	CisCharging	= true	ClastOperator
OcreateSound(pleaseDiscon.)	ClastOperator	= 4	ClastOperator
OwaitForAnswer(discon.)	ClastOperator	= 2	ClastOperator CisCharging

Table 5.2.: Assumptions and relevant changes of the defined basic operators

6. Experiments and results

The representations and functions developed within this thesis are combined in the proposed ILCA architecture in order to realize a Cognitive Technical System. The realized functionalities are inspired by human cognition and moreover designed and adjusted by experiments with a technical system interacting within a real world environment. In this context, especially the practical work with the technical system provides an intuitively access to the key features to be realized as learning, knowledge structuring, and complexity reduction. Based on the scenario presented in the previous section, the planning and learning capabilities of the cognitive architecture are exemplarily demonstrated by five experiments. These experiments include

1. generalization of knowledge by new relations,
2. application of meta operators,
3. complexity reduction and hierarchical planning,
4. consideration of new facts from interaction, and
5. refinement of the system's perceptual capabilities.

In the experiments, the described scenario is used in a complete version and in a reduced version as well. The reduced version of the scenario is limited to a certain part of the environment and used to demonstrate the generalization of knowledge. By generalization, the system is able to build and to refine a mental model automatically. Here, the cognitive function of planning can be further extended by the application of meta operators. In order to reduce the complexity of the complete scenario, the learned knowledge is represented in a hierarchical structure of action models with different degrees of abstraction. Based on that knowledge, context-sensitive action spaces are derived, which only contain the actually relevant information to generate goal-directed plans. In addition, the mental model can also be specified by new facts from interaction. By means of changes in the environment, the generalization of knowledge, or perceptual limits, the mental model may include certain ambiguities. Thus, the derived mental action space can contain conflicts, which may be solved temporarily by a short-term memory and persistently by the refinement of the system's perceptual focus.

In order to support the work with the robot-system and the cognitive architecture, a Graphical User Interface (GUI) is developed and applied. Here, the GUI (see Fig. 6.1) is used to test new cognitive functions, to control the connected technical system, to process and to visualize learned knowledge, and to monitor certain states of the system.

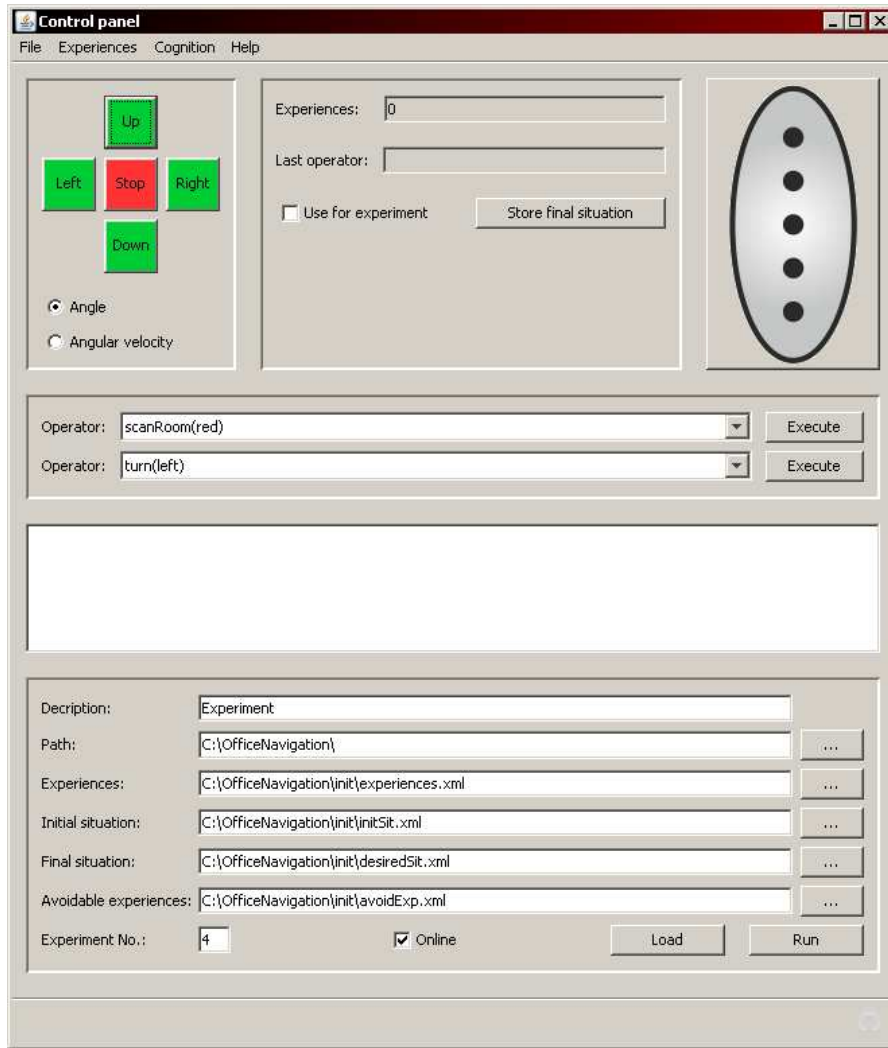


Figure 6.1.: Graphical User Interface to configure and to control the experiments

Furthermore, the experiments that are presented in the following sections can be configured and controlled by the GUI.

All quantitative and qualitative results presented in the following sections are measured from experiments with the mobile robot. Here, also the illustrated action spaces are measured snapshots from the system's knowledge and they are only edited in order to highlight certain aspects. Although the experiments are exclusively focused on the robot scenario, the presented results are also representative for similar systems. Neither the proposed approach nor its implementation by the cognitive architecture is specified to a certain kind of system or problem.

6.1. Generalization by new relations

In this experiment, the improvement of the system's behavior by learning from interaction with the environment is to be investigated. For this purpose, the number of wrong paths within the mental action space as well as the system's capability to generate and execute goal-directed plans are considered.

In order to illustrate the system's independence from its initial knowledge, which is an indication for a high degree of autonomy (see [RN03a]), only the number of available operators, the measured characteristics, and goals are defined in the cognitive architecture. Further initial knowledge or specific knowledge about the technical system to be controlled is not known by the system. Hence, the lack of information within the mental action space that is required to perform goal-directed behavior has to be learned automatically from interaction. In the considered scenario, the information to be learned involve also new relations (see Section 4.3.5) since the measured situation includes characteristics with numerical parameters.

Experimental set-up

The system gets two different goals, whereas only one is activated from the beginning. If an active goal is reached, this goal is deactivated and the other goal is activated. At the beginning of the experiment, the system has to perform a defined sequence of actions. Here, the correct sequence of actions leading the system directly to a goal is performed three times for every goal. Furthermore, all operators are also performed at those positions of the robot, where they do not change the situation or where they change the situation in a way which is not goal-directed. Hence, the first phase of the experiment corresponds to a structured exploration of the environment, in which the system performs 131 actions whose effects are stored as experiences. However, due to the fact that also characteristics with numerical parameters are focused, all situations differ from each other and the direct extraction of the operator's assumptions is prohibited. In the second phase of the experiment, several separate runs of interactions are started iteratively based on the 131 experiences. Thus, each run gets the same initial conditions and the expenditure of time for each run can be reduced significantly.

The different steps of each individual run are illustrated by the flow chart in Fig. 6.2. At the beginning, the experiences from the first phase of the experiment are used to build the action model and the operators' assumptions are generalized for the first time. After that, a mental action space is generated in order to plan a sequence of actions to the first goal. If this was successful, the first operator of the plan is selected. However, if the generation of a plan was not possible (e.g., no path between current situation and desired situation), the next goal-directed action is selected by a human operator. Alternatively, the system could select the action by a random-based strategy. Both strategies can be compared to the behavior of humans and they would finally induce similar re-

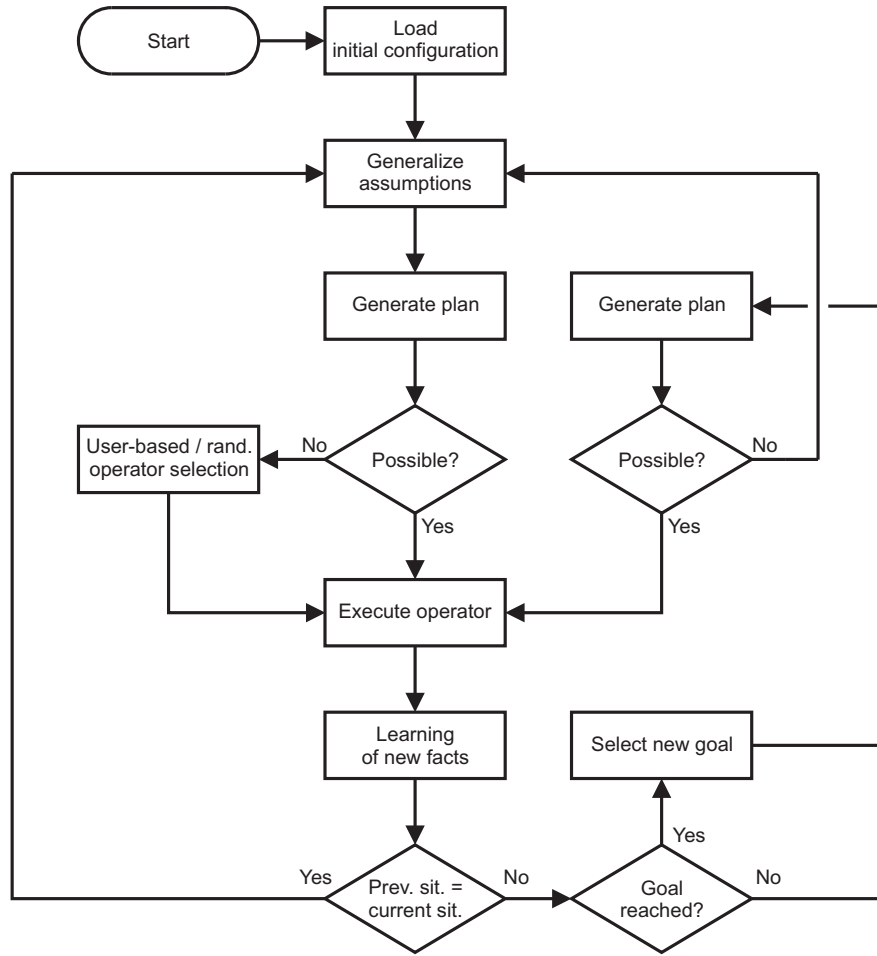


Figure 6.2.: Flow chart of the experiment

sults. Hence, the user-based selection is chosen in order to save time. After the next operator is selected (from a self-generated plan or from a human operator), the corresponding action is executed by the robot. The changes of the environment are stored as experience and used to refine the system's action model. If the current situation is equal to the previous situation, uncertainties in the action model are assumed. In this case, the operator's assumptions are generalized based on the refined action model in order to generate a new plan. Otherwise, if the situation changes after an operator was executed, the system checks whether the current situation corresponds to the current goal and selects a new goal if applicable. After that, a new plan is generated, independently whether a new goal was selected or not. If planning was successful, the next operator is executed and the lower loop repeats. However, if no suitable plan was found, the process of generalization is executed again. Hence, the user-based selection is not utilized until planning failed two times in series.

As mentioned above, only a subset of the defined operators is used in this experiment. The available operators are $O_{changeDirection}()$, $O_{turn}(left)$, $O_{driveInsideRoom}()$, $O_{driveOutsideRoom}()$, $O_{directToDoor}()$, $O_{gripObject}()$, and $O_{dropObject}()$. Hence, the whole interaction takes place in one room and the part of the corridor in front of that room. Therefore, the focused situation contains the characteristics $C_{insideRoom}$, $C_{relOrientation}$, $C_{accessibleAreaInFront}$, $C_{obstacleInFront}$, $C_{greenObjDet}$, $C_{objGripped}$, C_{xPos} , C_{yPos} , and $C_{orientation}$. In the first situation of the experiment, the robot stands inside the room and it is orientated to the opened door. Furthermore, a green bottle is placed behind the robot. The defined goals are $g_1 = (C_{insideRoom}=false, C_{objGripped}=true)$ and $g_2 = (C_{insideRoom}=false, C_{objGripped}=false)$, which is set to active initially. Hence, the robot drives between room and corridor (alternately, with and without a gripped bottle) if all assumptions and functions of the available operators are correctly known by the system.

By means of the generalization process, new relations are learned. These relations describe the dependency between a derived characteristic and one or several measured characteristics. As learning algorithm, here, Decision-Tree-Learning is applied. However, the same functionality could also be realized by other learning algorithms based on methods as Artificial Neural Networks or Support-Vector-Machines. The previous assumptions of an operator, which were used to learn the new relation, are replaced by the new derived characteristic. Hence, in the future, the operator can be applied also to situations which were not observed before. This is especially important in the used example scenario where the characteristics C_{xPos} , C_{yPos} , and $C_{orientation}$ with numerical parameters are contained in the focused situation. Hence, all situations differ at least regarding the parameters of these three characteristics. In Fig. 6.3, an example for a new relation learned from interaction is illustrated. Here, the derived characteristic de-

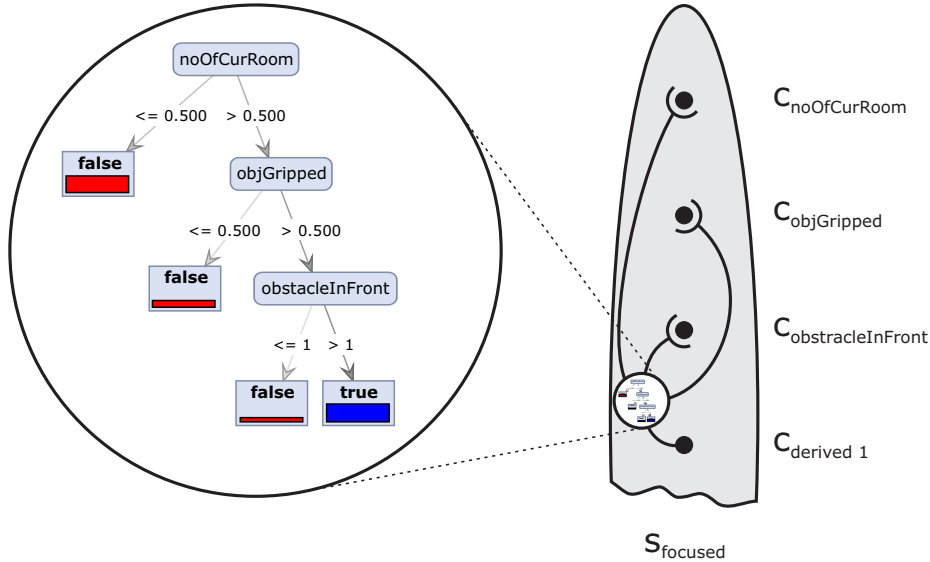


Figure 6.3.: Example for a new relation learned from interaction

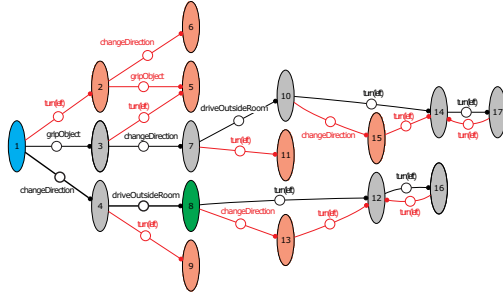
scribes whether the operator $o_{\text{dropObject}}()$ can be applied or not. The dependency between the derived characteristic and the measured characteristics $c_{\text{noOfCurRoom}}$, $c_{\text{objGripped}}$, and $c_{\text{obstacleInFront}}$ is represented by the decision tree contained in the relation as model. If the robot stands in front of a wall within the room and if an object is gripped, the operator $o_{\text{dropObject}}()$ can be applied to drop the object.

Experimental results

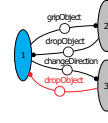
The more experiences are learned from interaction the more precisely the correct assumptions of the operators can be estimated. Especially at the beginning of each run, a lot of assumptions are not generalized correctly. Nevertheless, these wrong assumptions lead to hypothetic paths in the mental action space, which may be checked through real interaction with the environment. In Fig. 6.4, several mental action spaces are plotted, which were generated successively during a certain run. Between two successive action spaces, the robot performed a certain number of actions and the corresponding experiences were used to enhance the mental model, which can be seen by the decreasing number of wrong paths (red-colored).

In the situation s_1 of each mental action space, the robot stands in the room and it is orientated to the bottle. The active goal $g_2 = (c_{\text{insideRoom}}=\text{false}, c_{\text{objGripped}}=\text{false})$ is highlighted in the plots and it can be reached by the meta operator $o_{\text{meta}} = [o_{\text{changeDirection}}(), o_{\text{driveOutsideRoom}}()]$. This is possible in all of the plotted mental action spaces, except the one in Fig. 6.4(b). Moreover, it is remarkable that those mental action spaces generated in the early phase of the run contain a relatively large number of wrong paths. Nevertheless, these hypothetic paths also occur in the later phase of the run. For example, in Fig. 6.4(g), an additional wrong path is contained although the mental action space in Fig. 6.4(f) represents the interaction already more correct.

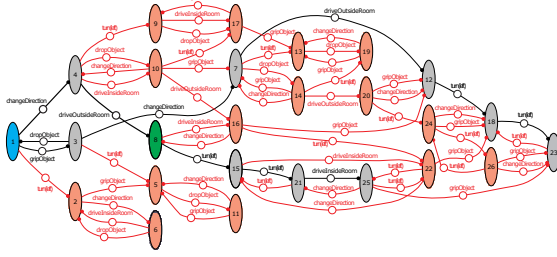
In order to compare the correctness of the mental models in different phases of the run, the mental action space after 26 actions (Fig. 6.4(a)) and the mental action space after 103 actions (Fig. 6.4(h)) are illustrated together in Fig. 6.5. The mental action space in Fig. 6.5(a) contains the correct path to the current goal, but ends after the operator $o_{\text{turn}}(\text{left})$ is applied the second time in series. Furthermore, the operators $o_{\text{driveInsideRoom}}()$ and $o_{\text{dropObject}}()$ are not contained. Hence, the mental action space can not be applied to plan longer paths, e.g., to the second goal. In addition to that, also a large number of wrong paths, which are actually not executable in the real world, are contained. This result from the fact that operators' assumptions are generalized indeed, but not very accurate due to a lack of sufficiently different experiences. In contrast to that, the mental action space in Fig. 6.5(b) can be applied to plan paths to both goals. The complete interaction is represented correctly except the operators $o_{10 \rightarrow 8} = o_{\text{turn}}(\text{left})$ and $o_{9 \rightarrow 7} = o_{\text{turn}}(\text{left})$. These wrong paths are still contained since they are not directed to any goal. Hence, the system had no inducement to check the paths during interaction.



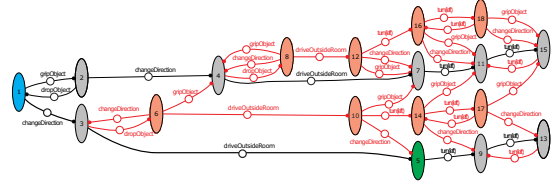
(a) Action space after 26 actions



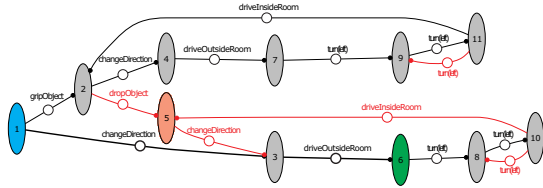
(b) Action space after 41 actions



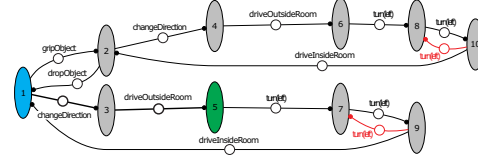
(c) Action space after 66 actions



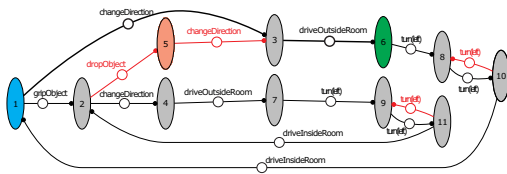
(d) Action space after 78 actions



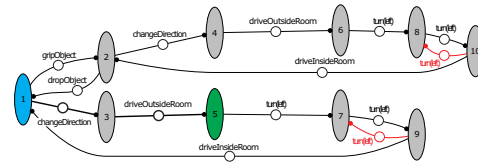
(e) Action space after 91 actions



(f) Action space after 103 actions



(g) Action space after 127 actions



(h) Action space after 139 actions

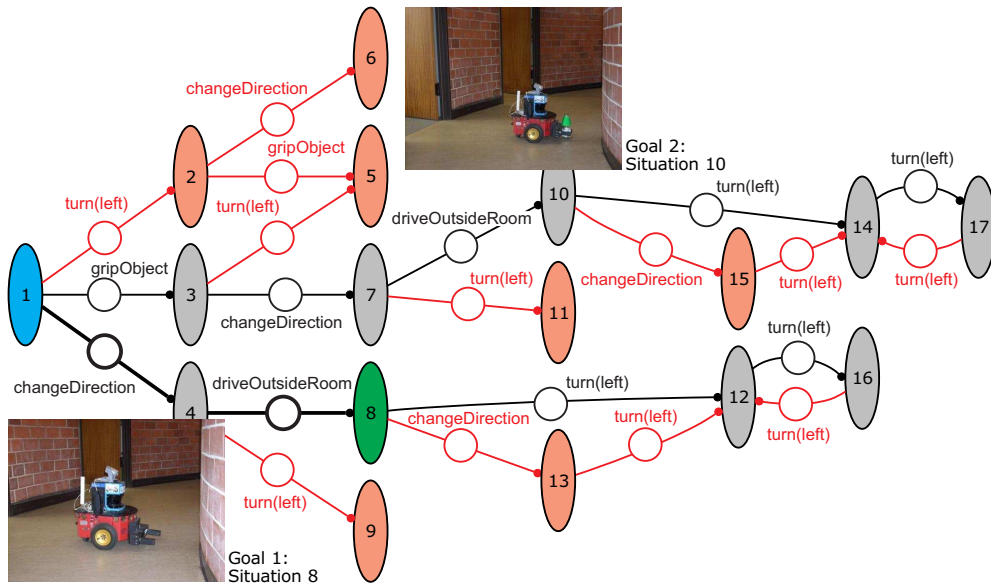
Figure 6.4.: Several mental action spaces generated during a run of the experiment

Within the second phase of the experiment, a total of three different runs with 180, 119, and 153 actions were accomplished. The runs were stopped after the robot planned

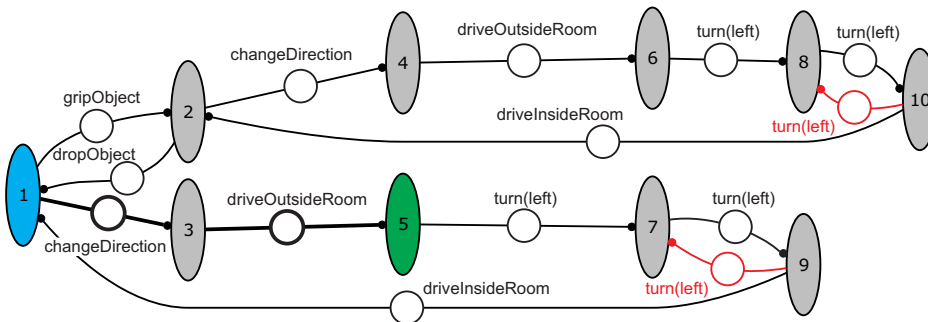
the path to each goal at least 12 times correctly. According to that, it is assumed that the systems will also generate exclusively correct plans in the future if the environment does not change. Before the execution of each action, the set of learned experiences and the current mental action space were stored. Furthermore, it was estimated

- how many actions were at least necessary to reach the current goal,
- whether the selection of the current operator was user-based, and
- whether the plan generated by the system was correct.

The fact whether a plan was generated correctly and the fact whether an operator was selected user-based is plotted in Fig. 6.6 over the number of all executed actions in each run of the experiment. Therefore, the arithmetical mean over all runs is calculated and



(a) Action space after 26 actions



(b) Action space after 103 actions

Figure 6.5.: Comparison of two mental action spaces regarding correctness

smoothed with a centered moving average of the order 11. According to the progression of the curves, the runs can be roughly divided into three different parts. In the first part of the run, the system is not able to generate a lot of plans correctly and requires the user-based selection of operators (or random-based selection alternatively) relatively often. Here, the peak at the beginning results from the fact that the path to the goal contained only one operator, which was correctly planned in each run. After a certain number of actions, the processions of the curves abruptly increase and decrease respectively. In this part the system learned how the situation changes if the robot drives from the office to the corridor and vice versa. Hence, the system is also able to generate longer plans by itself and the user-based selection becomes dispensable. From this time, the performance of the system is enhanced successively until all plans are generated correctly.

In order to illustrate the results of the experiment, the interaction is divided into several goal-related phases. A phase starts if a new goal is selected and ends if the selected goal is reached. According to this definition, the number of different events occurred between two goals is shown in Fig. 6.7 (except the first phase, which contains only one action). Here, the curves can be divided again into the three different parts mentioned above. The first diagram illustrates the number of executed actions and the number of user-selected operators. At the beginning, the system has to execute much more actions as required to reach the goal. This again changes abruptly so that the system is able to reach the goal in a minimum number of actions. However, as shown by the curve of number of user-selected operators and the curves in Fig. 6.7, further interaction is necessary until the mental model is accurate enough to generate all plans independently and correctly. In Fig. 6.7, the number of correct plans and the number of user-selected operators is plotted relatively regarding the number of actions within a goal-related phase.

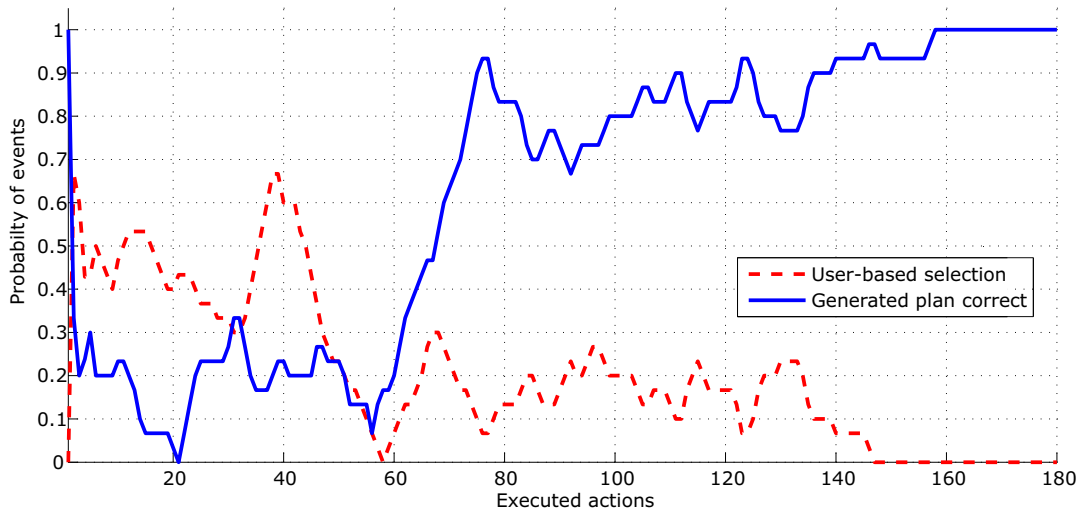


Figure 6.6.: Number of correct plans and user-selected operators

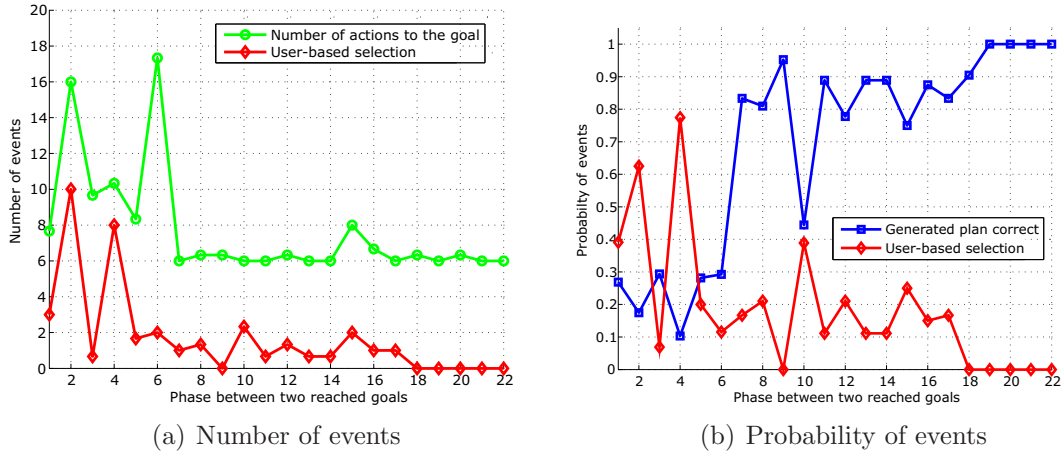


Figure 6.7.: Results regarding the phases between two reached goals

In addition to the analysis presented above, the interaction can also be sorted regarding the minimum number of actions which have to be executed to reach the current goal. Therefore, the results of all situations where the system had a certain distance to the goal are combined to one data set and sorted regarding the time of occurrence. In Fig. 6.8, the number of correct plans and the number of user-selected operators are plotted over the situations with the same minimal distance to the goal (smoothed with a centered moving average of the order 5). If the system requires only one action to reach the goal, the plan can be generated correctly from the very beginning. The larger the action-related distance between current situation and goal, the later the system can generate correct plans and does not require user-based operator selection.

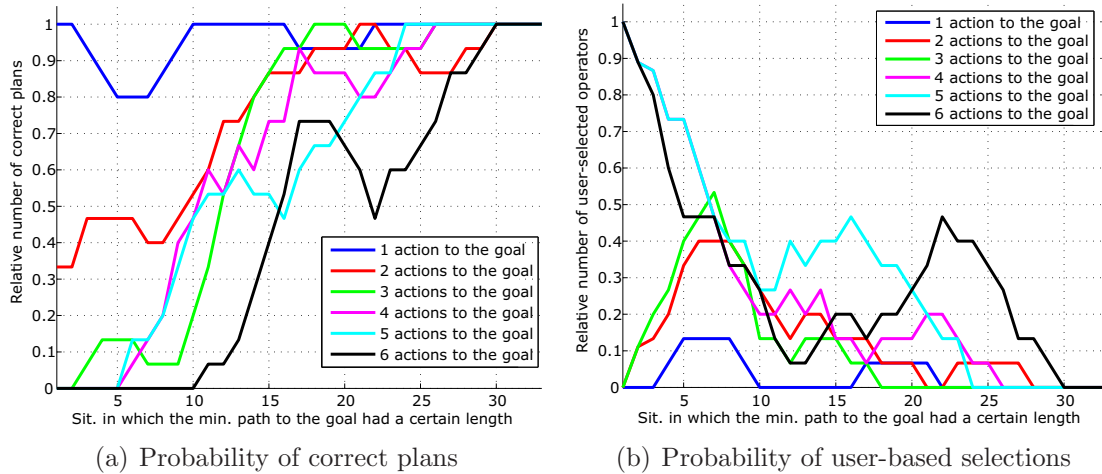


Figure 6.8.: Results of the analysis regarding the minimum path to the goal

6.2. Application of meta operators

The results of the first experiment show that the system is able to learn goal-directed behavior automatically and that the real world is mapped during the experiment more and more precisely in the system's mental model. Thus, the number of performed actions between two subgoals, the number of randomly selected actions (here, guided by a human), and the number of wrong plans decrease with the number of reached goals. However, due to the fact that a wrong plan can also be generated although a correct sequence of operators exists in the mental action space, these results might be enhanced if the planning process is optimized. Here, the application of meta operators (see Section 4.2.1) is a possible approach, which is presented in the following.

The problem is illustrated by the mental action space in Fig. 6.9. The mental action space was generated within a certain run of the first experiment between the 58th and 59th action. In situation s_1 , the robot stands in the corridor, grips the bottle and is oriented to the office. Hence, the system has the goal $g_2 = (c_{\text{insideRoom}}=\text{false}, c_{\text{objGripped}}=\text{false})$, which is a subsituation of the situations s_9 and s_{10} . However, only situation s_{10} can be reached by the correct meta operator $o_{1 \rightarrow 10} = [o_{\text{driveInsideRoom}}(), o_{\text{dropObject}}(), o_{\text{changeDirection}}(), o_{\text{driveOutsideRoom}}()]$. In contrast to that, situation s_9 can not be reached due to the fact that operator $o_{\text{driveOutsideRoom}}$ can actually not be applied to situation s_4 (robot is oriented to the door of the office).

If no additional information are available, the planning process selects the shortest path regarding the number of contained operators between the current situation and a goal situation. Furthermore, if several shortest paths exist, one of them is selected randomly. Hence, it can easily happen that a wrong path is selected although a correct

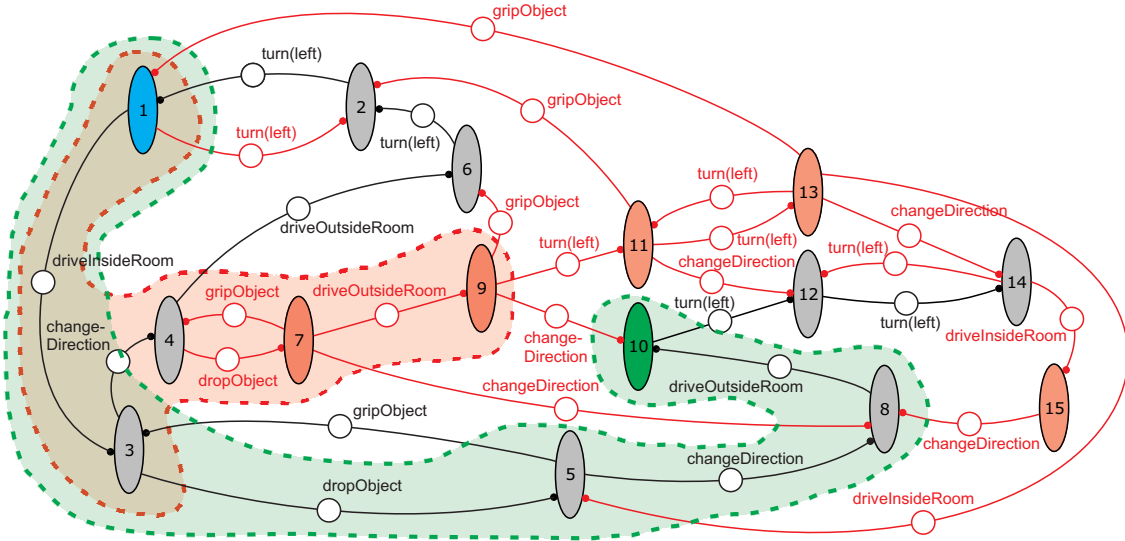


Figure 6.9.: Action space from the first experiment with wrong paths

alternative solution is also available (see above). In order to support the planning processes, meta operators represented by certain sequences of successful executed actions can be applied. Here, the meta operators can be compared with a set of available alternative paths in the mental action space. Based on the assumption that the order of goal-directed actions performed in the past is still goal-directed in the future interaction and/or in similar environments respectively, those paths of the current mental action space which correspond to the known meta operators are preferred.

Experimental set-up

In order to demonstrate the application of meta operators, two different runs of the first experiment are analyzed. Here, the meta operators represent sequences of actions which transfer the system directly from goal g_1 to goal g_2 and vice versa. For example, a possible meta operator is $o_{\text{meta}} = [o_{\text{turn(left)}}, o_{\text{turn(left)}}, o_{\text{driveInsideRoom()}}, o_{\text{dropObject()}}, o_{\text{changeDirection()}}, o_{\text{driveOutsideRoom()}}]$. Based on the defined meta operators, which could also be learned from interaction (see Section 4.3.3), it is checked whether the mental action spaces, which were used at the time when the system generated wrong plans, contain correct paths, too.

Experimental results

The results of the analysis are illustrated in Fig. 6.10. Here, the occurrence probability of different events (arithmetical mean over two runs) is plotted over the number of phases between two reached goals (see also Fig. 6.7). As presented in the previous

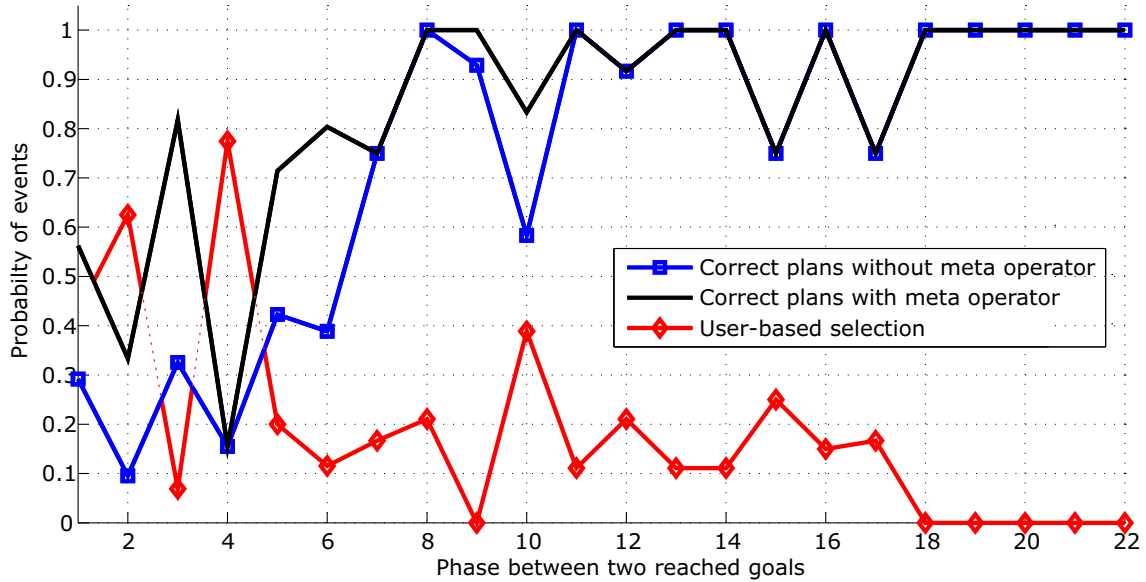


Figure 6.10.: Probability of different events during the experiment

section, the diagram contains the curves illustrating the number of user-selected operators and the number of correctly generated plans without the application of meta operators. Furthermore, also the number of potentially correct paths selected based on known meta operators is plotted. According to the curve progression, the application of meta operators may influence the interaction especially at the beginning of each run if the knowledge was generalized, but still characterized by a high degree of incorrectness. Hence, the probability to generate a correct plan can be increased by the application of meta operators, at least in the chosen example scenario.

6.3. Complexity reduction and hierarchical planning

The proposed system is able to learn the general assumptions of operators from several special examples, as shown in the first experiment. Hence, a correct prediction about an operator's function in a certain situation can also be made although the operator was not executed in exactly the same situation before. The experiment presented in this section starts with a correct and generalized action model in order to demonstrate an example of the system's capabilities to generate and execute plans which are based on a hierarchical structure of action models.

Experimental set-up

The system is provided with a set of already generalized experiences which allows the generation of correct plans from the beginning of the experiment. Furthermore, a set of experiences to be avoided is given. These user-defined operators, whose initial and final situations contain only a small number of characteristics, are used to remove undesired experiences (e.g., due to safety reasons) from the current mental action space before a plan is generated. Due to the fact that the meta action model is derived from the system's goals, the represented action-logic is structured in a context-sensitive manner.

At the beginning of the experiment, a meta action space is generated based on the available initial knowledge. After that and after each execution of an operator, the process of goal generation and selection is started in order to update the current goal if it is necessary. The current situation, the current goal, and the meta action space are used to derive a meta plan. According to the next operator of the meta plan, a local action space and a local plan respectively are estimated. Finally, the operators contained in the local plan are executed successively. If an unexpected situation occurs or if all operators of the local plan are executed, a new local action space is generated and the process repeats.

The initial knowledge defined by the system designer considers all operators described in the previous chapter and the related focused situation contains the characteristics $C_{\text{insideRoom}}$, $C_{\text{relOrientation}}$, $C_{\text{accessibleAreaInFront}}$, $C_{\text{obstacleInFront}}$, $C_{\text{greenObjDet}}$, $C_{\text{objGripped}}$, $C_{\text{noOfCurRoom}}$, $C_{\text{lastOperator}}$, $C_{\text{idOfDetTag}}$, $C_{\text{batteryStatus}}$, and $C_{\text{isCharging}}$. Hence, the robot can

interact with the whole environment including the corridor, the kitchen, and a variable number of offices. As operators to be avoided, the gripping of objects within the kitchen and the dropping of objects within the offices are defined. The defined goals are $g_1 = (c_{noOfCurRoom}=0, c_{objGripped}=true)$ with the priority of 2, $g_2 = (c_{noOfCurRoom}=0, c_{objGripped}=false)$ with the priority of 2, $g_3 = (c_{noOfCurRoom}=1, c_{charging}=true)$ with the priority of 3, and $g_4 = (c_{noOfCurRoom}=1, c_{charging}=true)$ with the priority of 1. Goal g_1 is activated at the beginning of the experiment, deactivated if the related situation is reached successfully, and reactivated if the characteristic $c_{isCharging}$ has the parameter true. Also goal g_2 is deactivated if it is reached successfully and activated if g_1 is reached. In contrast to that, g_3 is always activated, but due to its priority only selected if no other goal is active. Finally, the goal g_4 is activated if the characteristic $c_{batteryStatus}$ has the parameter 'empty' and it is deactivated if the parameter changed to 'middle'.

Experimental results

According to the described definitions, the robot can pick up an empty bottle in an office and it can place it in the kitchen. Moreover, the robot drives to the office in order to charge its batteries if the bottle is placed successfully in the kitchen or if the batteries are very low of charge ($c_{batteryStatus} = empty$). In Fig. 6.11, the entire action space and the related meta action space are shown for an environment with only one office (to keep the figure clear). The meta action space contains seven situations representing all possible combinations of the characteristics contained in the given goals. For example, situation s_6

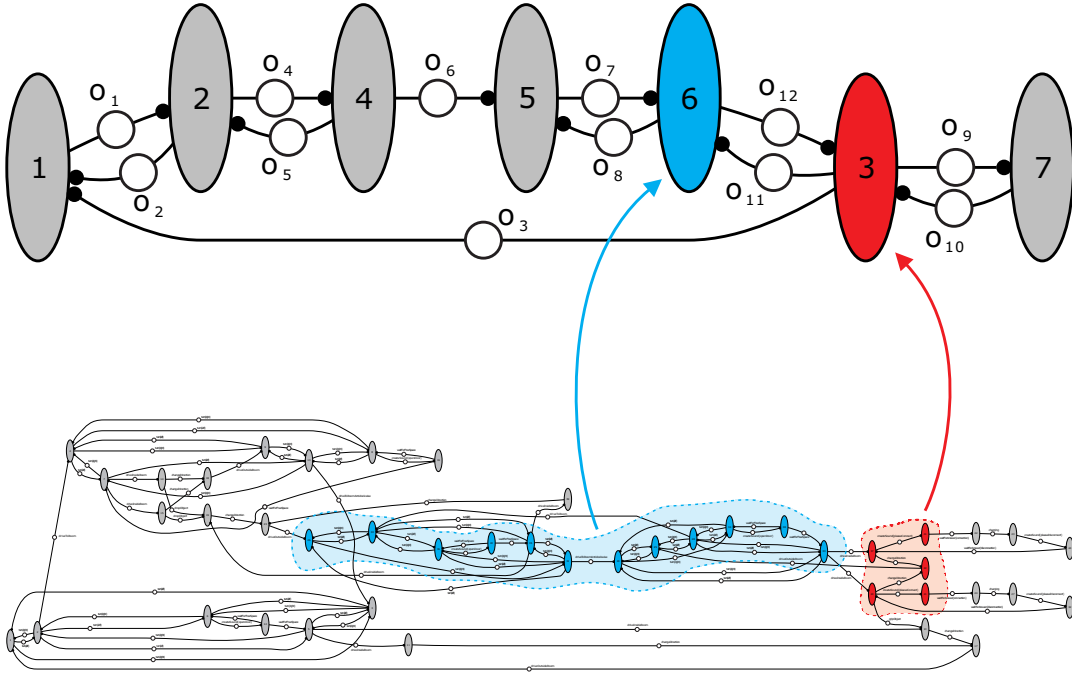


Figure 6.11.: Graphical illustration of the system's meta action space

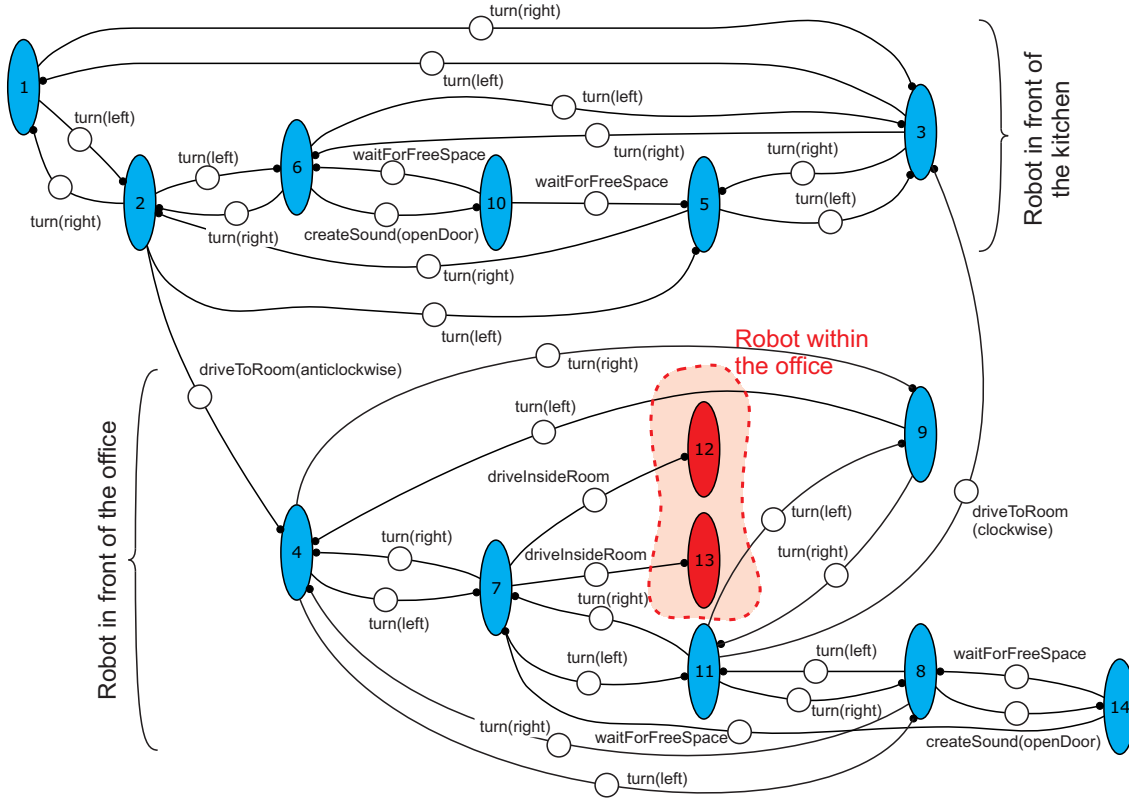


Figure 6.12.: Graphical illustration of the system's local action space

contains the characteristics $c_{noOfCurRoom}=0$, $c_{objGripped}=false$, and $c_{isCharging}=false$. This situation can be transferred by the operator o_{12} to situation s_3 with the characteristics $c_{noOfCurRoom}=1$, $c_{objGripped}=false$, and $c_{isCharging}=false$. The situations of the entire action space and the subsituations of the meta action space, which correspond to each other, are highlighted with the same color (red, blue, gray). Here, the situation s_6 (blue) and the situation s_3 (red) of the meta action space correspond to 12 situations and five situations respectively of the entire action space. As an example, the local action space corresponding to the operator o_{12} is illustrated in Fig. 6.12. This action space contains all 12 situations represented by the meta situation s_6 and the paths to the directly connected situations represented by the situation s_3 of the meta action space (in this case 2).

The presented experiment demonstrates an example for the proposed architecture's capability of hierarchical planning. Due to the fact that a generalized action model is already available, the system is able to generate a meta action space and several local action spaces in order to derive a shortest sequence of operators leading to the goal situation and subgoal situations respectively. Nevertheless, if an unexpected situation occurs, which happens regularly in real world environments, the current action space is no more valid and has to be rejected. Hence, a new action space is generated based on the new (previously unexpected) situation. In the experiment above, this effect typically

happens if the robot is within an office and tries to turn to the open door. Due to the fact that the shapes of all offices are more or less different, it can happen that the robot is not perfectly oriented to the open door after the operator $o_{\text{changeDirection}}()$ is executed. The resulting situation (with the characteristic $c_{\text{obstacleInFront}} = \text{close/between}$) is unexpected to the system since the described alternative function of the operator $o_{\text{changeDirection}}()$ is not contained in the provided initial knowledge. Now, the previously neglected operator $o_{\text{directToDoor}}()$ can be taken into account in order to generate a new local action space and plan.

6.4. Consideration of new facts from interaction

The experiment presented in the previous subsection illustrates the usage of the mental model in order to generate a goal-oriented sequence of actions. If the plan is executed in the real world, the system may transfer the current situation into a desired final situation. However, there are usually more than one alternative paths available and only one can be selected. In order to support the decision for technical systems, certain criteria, like the required time for execution, could be taken into account. Alternatively, also the required energy or the system's safety are suitable decision criteria. The related additional information (required time/energy and safety) can be logged during interaction (represented as weightings of the operators), but it give no evidence whether a whole (theoretically/mentally connected) sequence of operators is also executable successfully in the real world. Therefore, meta operators represented as a list of experiences can be used (see also Section 4.2.1). Meta-operators document the success of certain sequences of operators (at least for past interaction) and they can moreover be used to generate paths to standard goal-situations. Nevertheless, if the mental action space contains conflicts, the correct path matching to the real world can not be estimated by weightings and meta-operators.

Conflicts occur if the action model represents ambiguous information resulting from generalization, limited perceptual capabilities, or changes in the environment. Here, a certain operator may transfer one initial situation to several different final situations. In order to solve a conflict, this thesis proposes the refinement of the system's perception and/or the implementation of a short-term memory. If a conflict results from permanent changes in the environment, it can be helpful if the system's perceptual capabilities are refined (presented in the next section). In contrast to that, conflicts that are based on temporally changes in the environment might be solved by a short-term memory in order to enhance the performance of the system. For instance, the doors within the example scenario may be opened or closed (dependent or independent on a certain measurable fact), which is perceived by the robot as recently as it turns to the door. Here, it has to be avoided that the system recurrently plans to execute the operator $o_{\text{createSound}}(\text{openDoor})$ or the operators $o_{\text{turn}}(\text{left})$ and $o_{\text{turn}}(\text{right})$ alternatively.

Experimental set-up

The experiment corresponds nearly to the one presented in the previous subsection, but with the extension that the system learns additional facts from interaction. These additional facts include

- the average time required to execute the basic operators,
- successfully executed goal-oriented sequences of actions, and
- observed effects of all recently executed operators and meta operators.

The logged time is represented as weighting of the related operator. As soon as a weighting is measured for a certain experience, this information can be utilized to choose between different paths in the mental action space. If no weighting exists for a certain experience in the mental action space, this experience is weighted with the value 0. Hence, the applied Dijkstra-algorithm (e.g., see [Dij59]) tends to select all goal directed paths at least one time in order to update the corresponding weighting. All observed effects are stored as experiences in a short-term memory for a certain period of time. Thus, the currently right path in the mental action space regarding a certain conflict can be stored in order to avoid that the system tries to integrate one of the other alternative paths (involved in the same conflict) within a plan. Finally, successfully executed action sequences are represented as meta operators, but do not influence the behavior of the system in the chosen scenario. A suitable application of meta operators is presented in Section 6.2.

In this experiment, the environment of the robot is limited to the corridor, the kitchen and two offices. Apart from that, the scenario corresponds to the one of the last subsection. Hence, only the set of experience to be avoided and the experiences regarding the operators $o_{\text{driveToRoom}}(\text{clockwise})$ and $o_{\text{driveToRoom}}(\text{anticlockwise})$ have to be adapted. Consequently, the robot drives to the offices, collects empty bottles, carries them to the kitchen, and charges the batteries periodically. During the experiment, the doors of the offices are not opened the whole time and will also not be opened at all times when the robot asks for that. Furthermore, the robot may not find every time a bottle if it drives inside an office. In order to handle these situations, the mentioned short-term memory, whose dwell time is set to two minutes, is necessary. Furthermore, due to the fact that the system plans based on an action space, which is weighted regarding the operator's time for execution, the corresponding influence to the robot's behavior can also be observed in the experiment.

Experimental results

In Fig. 6.13, the system's meta action space is illustrated. In the situation s_1 , the robot stands in office 1 and its gripper is opened. The next goal corresponds to situation s_9 , which can be reached by the red-colored path. Thus, the robot grips the object (o_1), drives to the corridor (o_2), and drives inside the kitchen (o_{14}). Here, each operator

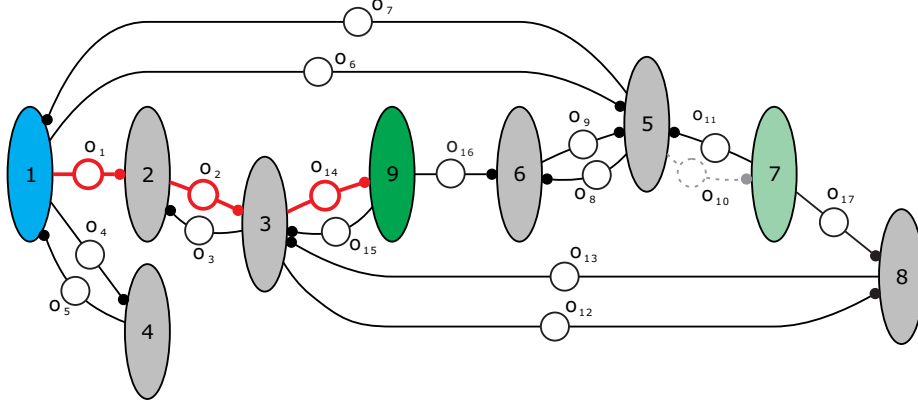


Figure 6.13.: Meta action space of an environment with two offices

corresponds to a separate local action space. The local action spaces related to the operators o_2 and o_{10} are shown in the Fig. 6.14 and 6.15 in order to illustrate the usage of weightings, meta operators, and short-term memory.

The local action space in Fig. 6.14 corresponds to the interaction of the robot on the corridor before office 1 (operator o_2) and it is used to illustrate the learning and the usage of weightings. Every time the robot performs a certain operator, the time required for execution is stored in a list of experiences. Then these experiences are used to weight the mental action space. Due to the fact that the weightings are assumed as zero from the beginning of the experiment, the robot explores automatically different paths until the one with the lowest expenditure of time is found. In the situation s_1 of the mental action space in Fig 6.14, the robot stands on the corridor, grips a bottle, and is oriented to the wall (after the robot performed the operator $o_{driveOutsideRoom}()$). Due to the fact that the system has no information about the operator's weightings at this time, the shortest path regarding the number of operators is chosen. Hence, the system plans the execution of the green-colored path $o_{turn}(\text{left})$, $o_{driveToRoom}(\text{clockwise})$, $o_{turn}(\text{left})$, and $o_{driveInsideRoom}()$ in order to reach the current goal $g_1 = (c_{noOfCurRoom}=0, c_{objGripped}=\text{true})$. If this was successful a corresponding list of experiences is stored to describe the operator o_2 of the meta action space. Hence, this meta operator can be chosen instead of planning in the local actions space to transfer the robot from the situation s_2 to the situation s_3 of the meta action space. Besides the meta operator also the weightings of the contained basic operators are stored. Hence, the system plans a different path the next time if the robot is in the same situation (s_1 of the local action space) and the same goal (g_1) is aimed. However, after the meta operator $o_{meta} = [o_{turn}(\text{right}), o_{turn}(\text{right}), o_{turn}(\text{right}), o_{driveToRoom}(\text{clockwise}), \dots]$ is executed, the system is able to recognize that this path is much longer regarding the expenditure of time and it will chose the shorter path in the future. Nevertheless, this interaction was useful to explore different goal-directed paths.

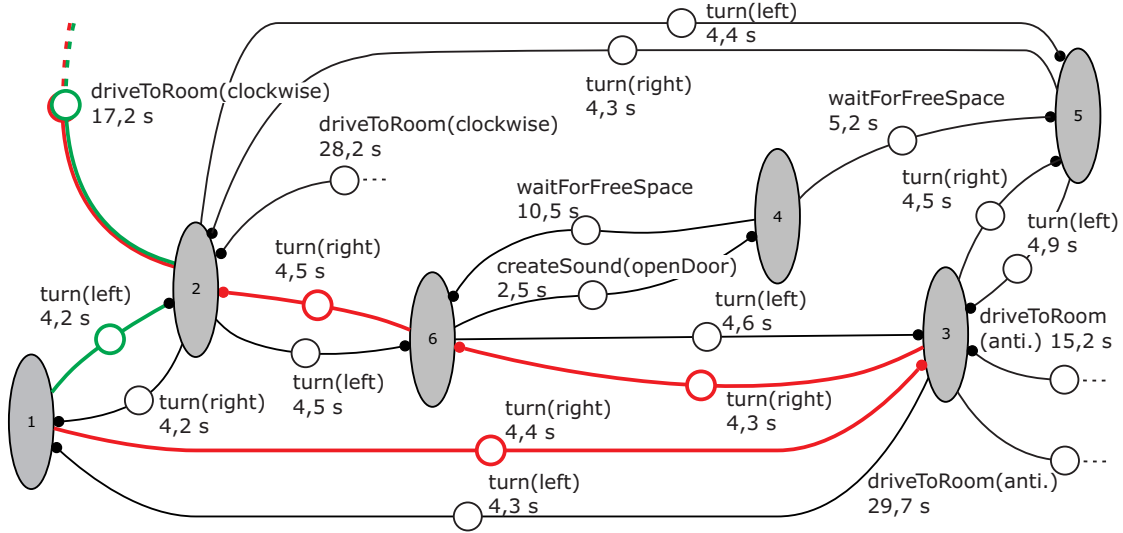


Figure 6.14.: Local action space related to the operator o_2 of the meta action space

The usage of short-term memory is illustrated by the local action space in Fig. 6.14, which corresponds to the operator o_{10} of the meta action space in Fig. 6.13. In situation s_1 , the robot stands in front of the office 2, it has no object gripped, and it is directed to a closed door. In order to reach the current goal $g_1 = (c_{noOfCurRoom}=0, c_{objGripped}=true)$, the next subgoal corresponds to the situation s_8 . Accordingly, the system executes the operators $o_{createSound(openDoor)}$, $o_{waitForFreeSpace}()$, $o_{turn(left)}$, and $o_{turn(right)}$. However, due to existing conflicts, the operators $o_{waitForFreeSpace}()$ and $(o_{turn(right)})$ do not change the situations in the desirable goal-directed way. This is also stored in the system's short-term memory illustrated in the lower part of Fig. 6.15. After the operator $(o_{turn(right)})$ is executed the second time, the system recognizes that the aimed subgoal is currently not reachable. Hence, also a new meta plan has to be generated to the current goal g_1 in order to exchange the subgoals.

The presented example describes the utilization of a short term-memory, but its benefit becomes not apparent until the experiment is accomplished with a dwell time of zero. Here, the system would lose the information that the door of the office 2 was not opened after the operator $o_{waitForFreeSpace}()$ was performed. Hence, the system would plan and execute the operator again and again. The same effect can also be observed if the robot drives into an office where no bottle is placed. In this case, the robot would plan and execute the same sequence of operators $(o_{changeDirection}(), o_{driveOutsideRoom}(), o_{turn(left)}, o_{turn(left)}, o_{driveInsideRoom}())$ iteratively until a bottle is placed at the defined position.

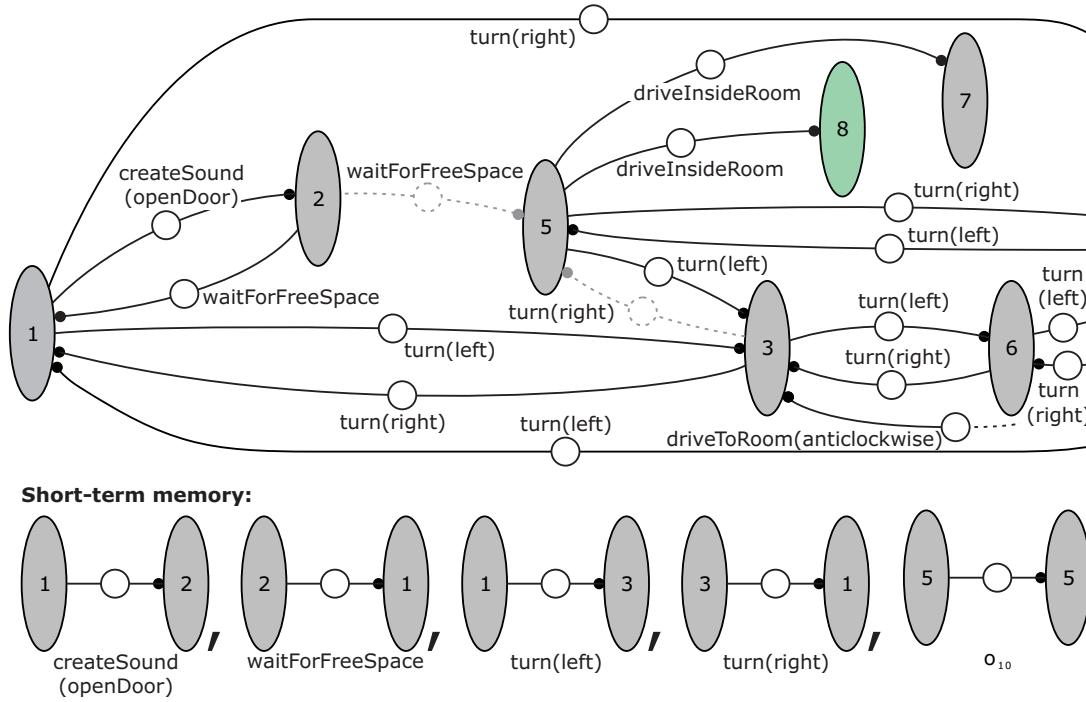


Figure 6.15.: Local action space related to the operator o_{10} of the meta action space

6.5. Refinement of perceptual capabilities

In this thesis, conflicts are considered as ambiguous representations in the system's mental model. Such a conflict exists if more than one function of a certain operator is known (from previous observations). One strategy to solve these conflicts is the usage of a short-term memory as presented in the previous subsection. Thus, the system stores recently gathered facts from interaction in order to estimate the temporally correct functions of those operators which are represented uncertainly. However, in most of the cases, this strategy does not consider the roots of the existing conflicts: an incomplete and/or wrong focused perception of the system. E.g., if the relevant details of two different scenes are not considered (or can not be measured), they are represented by the same situation. In this case, two different effects of a certain operator may be observed if this operator is applied to the different scenes which are represented equally. Hence, a cognitive system should also be able to refine its perception in order to understand the behavior of its environment (which may moreover change during runtime) fundamentally.

Experimental set-up

In common with the system in the previous experiment, here, the system gets also a set of experiences as initial knowledge and interacts with the environment according to the defined goal situations and undesired experiences. Furthermore, a short-term memory is used to store the results of each performed operator for a certain period of time.

In addition, if an experience corresponding to a conflict occurs, the system stores the measured characteristics of the initial situation and tries to find a previously not considered characteristic which can be used to distinguish initial situations unambiguously. Therefore, the measured characteristics of all initial situations involved in a conflict are analyzed and weighted (between 0.0 - 1.0) by a correlation-based weighting algorithm (see [Rap10]). If a characteristic with a nominal parameter gets the weight 1.0, the characteristic can solve the conflict and is added to the system's perceptual focus in order to refine the knowledge about the operators' assumptions and avoid the conflict in the future.

In this experiment, the same scenario is used as in the previous section, except that the goals g_3 and g_4 , which are related to the charging of the batteries, are permanently deactivated. Accordingly, the characteristics $c_{\text{insideRoom}}$, $c_{\text{relOrientation}}$, $c_{\text{accessibleAreaInFront}}$, $c_{\text{obstacleInFront}}$, $c_{\text{greenObjDet}}$, $c_{\text{objGripped}}$, $c_{\text{noOfCurRoom}}$, $c_{\text{lastOperator}}$, $c_{\text{idOfDetTag}}$, $c_{\text{batteryStatus}}$, and $c_{\text{isCharging}}$ are focused. Furthermore, these characteristics and all other ones (see Section 5.2.3) are contained in the measured situation. In order to handle temporary changes in the environment, like the opening and closing of doors, a short-term memory with a dwell time of two minutes is used. However, the learning of weightings and meta operators is not considered due to the fact that both representations can not be used to solve conflicts. As an additional arrangement, closed doors of unoccupied offices are marked with a red sticker to give the robot the information that the door is not opened if it asks for that. However, this is not represented in the system's mental model and has to be learned from interaction. Similar arrangements or changes in the environment can also be done during runtime. Thus, the humans can place signs and the system has to derive their meanings automatically as a certain type of communication between human and technical system. According to the defined conditions, the robot drives to the offices and tries to carry empty bottles to the kitchen. If an office door keeps closed or if an office does not contain an empty bottle, the system changes its plan and tries to reach the given goal by an alternative sequence of actions. As soon as the system perceives an experience which belongs to a known conflict, the system tries to solve this conflict.

Experimental results

The local action space illustrated in Fig. 6.16 already contains several conflicts. As indicated by the situation in the upper left corner, the robot stands on the corridor, grips no object, and it is oriented to a closed door of office 2. Due to the conflicts, the system is uncertain about whether the door of the office is opened after the operators $o_{\text{createSound}}(\text{openDoor})$ and $o_{\text{waitForFreeSpace}}()$ are performed. Furthermore, the office door can be opened or closed if the robot orients itself to the office by the operators $o_{\text{turn}}(\text{left})$ and $o_{\text{turn}}(\text{right})$ respectively. Finally, an empty bottle can be within an office or not. However, that is not realized by the system until the operator $o_{\text{driveInsideRoom}}()$ was executed. As described in the previous section, all observed alternative experiences related

to these conflicts are stored in the system's short-term memory. Hence, the mental model is adapted to the current circumstances in the environment and may be used to generate action spaces without conflicts. However, this action space may only be valid for a certain period of time due to the fact that the environment is changing continuously.

By the experiment, the permanent solution of conflicts through the adaptation of the system's perceptual capabilities is shown. However, this requires that suitable measured characteristics are available. During the experiment, no bottle is placed inside the offices. Hence, the robot drives alternately between both offices. The door of the office 2 is always closed and it is only opened if someone is within the office. If this is not the case, a red sign is placed at the door. Although the red color is detected by the robot and represented by the characteristics $c_{\text{redObjDet}}$ and $o_{\text{posOfRedObj}}$, this does not influence the system's behavior due to the fact that the characteristics are not considered in the focused situation.

The first time the robot drove to office 2, nobody was inside the office. However, due to a lack of enough example situations, the solution of the corresponding conflict were not executable. When the robot drove to office 2 the second time, someone was inside and opened the door after the robot asked for that. After that, the system had two different example situations (one for each observed function of the operator $o_{\text{waitForFreeSpace}}()$) and started the process of conflict resolution. Here, the only characteristic, which was weighted with 1.0, was the characteristic c_{xPos} . However, due to the fact that this characteristic has a numerical parameter, the conflict can still not be solved. Finally, if the robot drove the third time to office 2, nobody was inside again. By the additional example situation, the characteristics $c_{\text{redObjDet}}$ and $c_{\text{posOfRedObj}}$ were weighted with 1.0 and suitable to solve the conflict permanently.

In order to solve the conflicts in the mental action space, all operator nets of the operator $o_{\text{waitForFreeSpace}}()$ are deleted from the systems action model. After that, those experiences which were collected to solve the conflict are integrated in the action model.

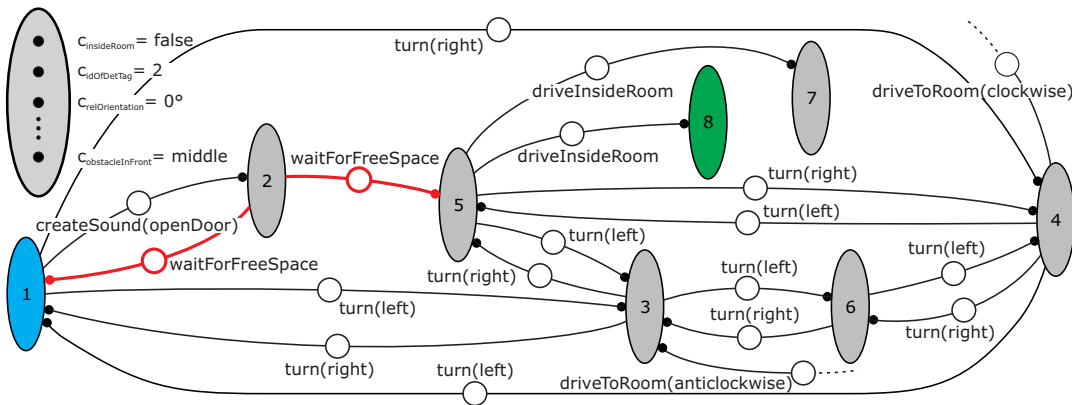


Figure 6.16.: Local action space with several conflicts

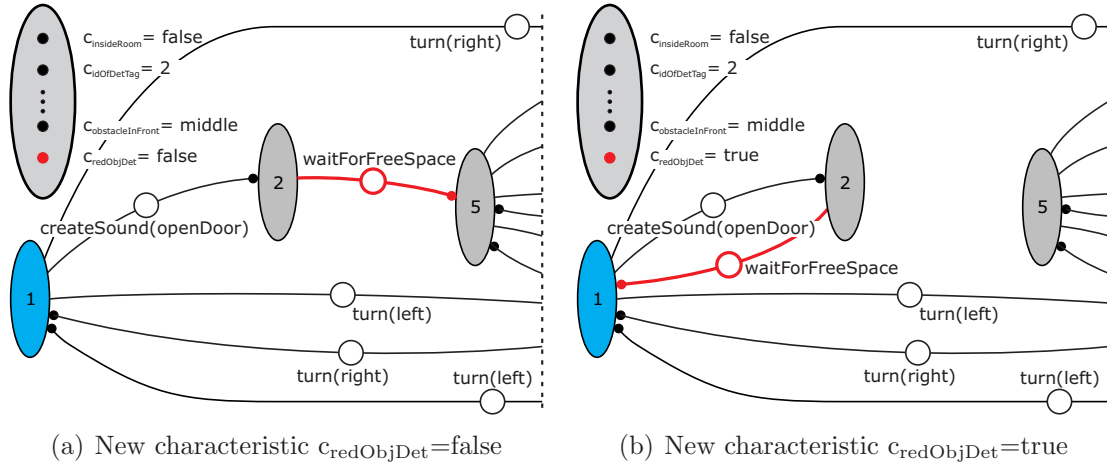


Figure 6.17.: Local action space after conflict resolution

Therefore, the characteristics of the initial and final situations contained in these experiences are limited to the previously focused characteristics and the characteristics $c_{redObjDet}$ and $c_{posOfRedObj}$. Furthermore, the two new characteristics are also included in the focused situation. By means of the described changes, the system may consider two different local action spaces if it stands in front of a closed door, which is illustrated in Fig. 6.17. In situation s_1 of Fig. 6.17(a), the red sign is not detected by the system. Hence, the system assumes based on the refined mental model that the door of the office will be opened after the operators $o_{createSound(openDoor)}$ and $o_{waitForFreeSpace}()$ were performed. In contrast to that, the system assumes that the door is not opened if it perceives a red object (see Fig. 6.17(b)). In this case, the system does not perform the operator $o_{createSound(openDoor)}$, but it generates an alternative plan directly.

In the experiment, the conflict regarding the operator $o_{waitForFreeSpace}()$ was resolved persistently. However, through further interaction with the environment, another conflict is extended. Now, the system can perceive an open door, a closed door without red sign, and newly a closed door with red sign if it turns to an office. Candidates for the solution of that conflict may be the characteristics $c_{obstacleLeft}$, $c_{obstacleRight}$, $c_{timeOfDay}$, and/or $c_{idOfDetTag}$. Nevertheless, the mental action space of a system in a dynamical environment can never be completely free of conflicts due to the fact that it is impossible to predict all events (also true for humans). However, this does not mean that the behavior of the system is not rational.

6.6. Methodical conclusions

In the previous subsections, the functionalities of the developed cognitive functions and related representations are presented by five experiments with a mobile robot as an example system. Here, qualitative as well as quantitative results illustrate the performance of the system. In addition to that, this section presents several methodical conclusions, which have to be considered in the overall context of the five experiments.

The experiments show that the proposed learning functions enable a system to obtain goal-directed behavior although no initial knowledge or system-specific settings are provided to the cognitive architecture. Furthermore, the system's behavior is enhanced (e.g., less wrong paths in the mental action space) with the number of perceived experiences, but the system keeps flexible in order to consider new facts and unexpected changes in the environment. Especially, the first experiment demonstrates that the system's knowledge seems to change stepwise. In this case, the learning behavior can be divided roughly into three different phases. However, this effect might also result from the specific experimental set-up and it will probably vary with different scenarios or technical systems.

According to the experiments, it has to be emphasized that a single learning mechanism or a single kind of knowledge representation is not enough to build and to structure a mental model mapping the real world sufficiently. Only the interplay among the different cognitive functions and related representations, which are all based on the same methodical background, allows the realization of a flexible system which is capable to interact goal-directed within a complex environment. Also the combination of all functions within one algorithm *a posteriori* would not be helpful since in this case the further development of the approach would be limited strongly. If the proposed cognitive architecture is applied to other systems, additional cognitive functions or variants of the existing cognitive functions (e.g., with a different search algorithm etc.) might be useful. However, the extension of the architecture is explicitly intended and prepared.

As exemplarily illustrated by the experiments with the mobile robot, the proposed knowledge structure combines all representations, which are used and modified by the different cognitive functions. Especially, the function of perception, which is also based on learned knowledge, plays a central role due to the fact that it is used to process situations from the real and mental world as well. Without a common methodical background, like the one used in this thesis, the realization of an efficient interplay among all cognitive functions and the integration of new mechanisms is more difficult and less flexible.

The qualitative results of the experiments are visualized according to the symbolism coming with the SOM approach. By means of the graphical illustration of the system's knowledge (e.g., the relations among characteristics or the current mental actions space), which is the basis of decision-making, the system's behavior can be tracked and ana-

lyzed in an intuitive manner. Moreover, the visualization is an essential element for the purpose of the design and the test of new cognitive functions.

Finally, it becomes apparent that the realization of a Cognitive Technical System requires experiments with a real technical system interacting in a dynamical environment at an early stage of development. Experiments which are based on simulated world environments may help to design some basic functionalities, but they are not suitable to design cognitive functions to be implemented in a technical system. In this regard, the development of cognitive functions, which are used to handle uncertain and/or noisy sensor measurements, is a typical example. Moreover, applications with mobile systems, like the chosen robot, help to draw analogies between the technical system's interaction and human behaviors.

7. Summary and future work

This thesis presents the realization of Cognitive Technical Systems by means of the development of a novel framework which is based on Situation-Operator-Modeling. The resulting main achievements and related benefits are summarized in the next subsection. In addition, new ideas and alternative applications, which were identified in conjunction with this thesis, are listed at the end of this chapter, each with a brief description.

7.1. Summary

In previous work, the Situation-Operator-Modeling approach [Söf01c] and the first implementation of a SOM-based cognitive architecture [Ahl07] were presented. Based on that research, Petri net patterns were developed to simulate and to analyze SOM-based models [Gam06, GOS07] in an intuitive and flexible manner.

The developed SOM-based Petri net patterns and the closely related state space analysis technique offer the perfect basis for modeling and simulation of human interaction and cognition. In order to develop a SOM-based representational level for technical systems, the Petri nets patterns are used to design a framework modeling the functions and procedures of the human mind. Here, especially learning plays a central role for the realization of autonomous behavior. Different memories and cognitive functions are combined within a cognitive architecture, which is used to control a mobile robot within a dynamical environment.

The successful completion of each step described above builds an important milestone in order to realize a Cognitive Technical System. Furthermore, the developed approaches exhibit also additional benefits for alternative purposes and applications. In the following, the main achievements of this thesis are given in detail.

- As a first step, the implementation of SOM-based models through patterns of high-level Petri nets was considered. Here, an arcade game application was used as an example micro world with human interaction. The interaction was represented by state space patterns, which were used to generate an action space corresponding to a discrete state space representation with situations as states and operators as state transitions. As an example, the action space was analyzed by formal state space queries in order to detect human errors automatically. For systems with a higher degree of complexity, the complexity was reduced by the usage of partial action spaces focusing only the relevant aspects of interaction. Based on that research, adaptable Petri nets patterns were developed, which can also be changed

during interaction. Finally, the represented interaction can also be automatically visualized and converted to a general exchange format.

Through the developed methodology, the interaction of arbitrary systems of humans and technical systems can be represented, analyzed, and visualized.

- The high-level Petri net patterns were used to develop elements as well as related cognitive architectures to realize a flexible and adaptive knowledge representation for technical systems in order to realize autonomous behavior. Therefore, the SOM approach was utilized as an intermediate level between real world's phenomena and technical model. Based on the action space as central problem description, a framework of several cognitive functions and representations were developed in order to model human capabilities as learning, planning, and complexity reduction. Thus, it was possible to design a novel kind of hierarchical knowledge representation, which can be restructured from interaction with the environment. In this regard, different learning functions were developed, which are also applicable in real world scenarios. These learning functions realize the integration of new facts from interaction, the generalization of knowledge from special observations, the consideration of temporal dependencies, the simplification of complex action spaces by abstraction, and the resolution of uncertainties by the adaptation of perception.

The developed framework, which can be extended easily by other KI methods, can be used to design integrated cognitive architectures for different purposes.

- The representations and functions of the developed framework were combined within a cognitive architecture. The architecture coordinates the execution of cognitive functions and it is structured into three levels for skill-based, rule-based, and knowledge-based behavior. As the basis for planning, a context-sensitive action space (working memory) is derived from the recently perceived experiences (short-term memory) and the system's general knowledge (long-term memory). If planning fails, the system's knowledge is refined by processes of deduction and induction. Furthermore, the knowledge is extended by new facts learned from interaction. The cognitive architecture itself is completely independent from specific problems or applications. Here, only the measured characteristics and available operators have to be named. In addition, also goals, actions to be avoided, and initial knowledge can be defined.

The cognitive architecture can be connected through a network connection with different kinds of technical systems in order to provide cognitive control or cooperate with them.

- The development of the cognitive architecture was attended by experiments with a mobile robot interacting in a dynamical office environment. In contrast to virtual or laboratory environments, here, it was necessary to consider aspects as safety (the robot may not drive against walls or humans), erroneous measurements, and changes of the environment (especially caused by humans), which significantly influenced the developed functions. Therefore, an existing Pioneer 3DX platform had

to be adapted by additional software and hardware components. With the robot as experimental platform and the chosen scenario, it was possible to demonstrate that the developed methodology can be applied actually to complex technical systems. In this regard, qualitative as well as quantitative results, which illustrate the learning behavior of the system, were measured.

The extended robot system provides an ideal experimental platform for future research on Cognitive Technical Systems.

Without formalisms, the complexity of the real world is not comprehensible and can not be described in a unified manner. However, formalisms are limited and may not describe all relations sufficiently. Hence, an intermediate level between real world and technical model is necessary.

This thesis refines the approach of a structured intermediate level between real world and technical model, which is unique and not comparable to other existing approaches. Based on the work of [Söf01c] and [Ahl07], this thesis contributes an extensive description of human cognition by Situation-Operator-Modeling and adds crucial details as the automatic adaptation of perception, complexity reduction by differentiation among short-term, long-term, and working memory, the hierarchical structure of action spaces with different degrees of abstraction, the automatic selection of goals, and learning functions which are applicable in real world scenarios. In this regard, the novel kind of knowledge structuring, which is developed and tested in this thesis has a strong benefit, also for other scientific fields. The proposed models and mechanisms were developed through experiments with a real mobile robot, but they can also be applied to arbitrary kinds of technical systems.

7.2. Future work

During the research of this thesis, which was primarily focused on the realization of a Cognitive Technical System, several alternative application fields and ideas for methodical extensions were identified. The most promising ones are listed below in order to indicate possible future work based on this thesis.

- Besides the presented scenario, the cognitive architecture can also be used in combination with other systems in different applications. This is especially supported by the underlying generic concept and its system-independent implementation. Furthermore, a standardized communication interface is provided to connect different systems and to exchange information easily. Possible applications would be the cognitive control of technical systems (as demonstrated with the robot), the cooperation with other cognitive systems, and the assistance of humans (see [GS07, FGM⁺09]). For Human-Machine-Systems in particular, it can be helpful if technical systems can behave like humans and understand human

behavior respectively. However, in which way the communication and interaction between humans and Cognitive Technical Systems (based on the proposed cognitive architecture) have to be designed, has to be investigated in the future.

- If the cognitive architecture is applied within other contexts, novel kind of challenges might arise, which were not focused within this thesis. However, due to the fact that the cognitive architecture is only a special instance for a combined system of the underlying framework's representations and functions, the architecture can be modified, extended, and completely revised easily. In this regard, this thesis provides a comfortable starting point for further investigation regarding the proposed approach and its technical implementation. For example, the developed cognitive functions could be extended and improved. Here, especially the implemented algorithms for the generation of action sequences and the analysis of measured data can be simply exchanged.
- In addition to the already existing models, which represent general knowledge or rules, the system could be extended by other representations. Thus, completely new research directions can be focused. The additional representation may be learned and processed within the main cognitive cycle of the architecture directly or through secondary processes. As an example, the system might learn safety critical information about situations and/or operators from the interaction with the environment or the communication with other systems. Currently, those aspects are considered by weightings in the action model and mental action space respectively (see also [EGVS10]).
- In the presented cognitive architecture, two modules are used to generate and select goals. However, here, the generation is limited to the activation and deactivation of goals. Furthermore, the selection of goals is exclusively based on the active goals' priorities and activation times. The next level of development could be the learning of goals from interaction. In this regard, goals are derived from certain 'primary drives' like the need for energy, etc. (see [Hul52]). After that, these goals may be further abstracted according to the existing structure of the systems knowledge. Hence, the influence of emotions and drives to a system's behavior may be investigated and the advantages and disadvantages for different systems and applications can be judged. In addition, the selection of goals can be based on the current mental action space in order to optimize the sequence of reached subgoals. The described functionalities can be based on the proposed framework, which already provides the required basic representations and functions.
- At a connected technical system, the basic actions and measured characteristics are defined by hard-coded controller and filter routines. Based on the available sensor measurements, the proposed cognitive architecture is able to learn abstract rules from interaction in order to enhance its planning and perception capabilities. This is realized exclusively within the cognitive architecture independently from a certain technical system. However, this leads also to the fact that learning of

sensomotoric behaviors and reflexes is not focused by this thesis. Nevertheless, the cognitive architecture also allows the addition of new operators and measured characteristics during runtime. Like the learning of skills in a biological system by a large number of iterations, also a technical system could be directly equipped with pattern recognition algorithms and adaptive controllers to provide new measured characteristics and operators to the cognitive architecture. Thus, the autonomy of the combined system could be increased significantly.

- In order to solve complex problems, also several instances of the cognitive architectures can be applied together. Here, the architectures could work on the same level or might be connected in a hierarchical and heterogeneous (architectures with different tasks) structure. In any case, this is provided by the proposed framework, which enables the definition of architectures as class objects with suitable interfaces. Hence, several similar cognitive architectures can communicate with each other or process certain models collaboratively. Nevertheless, whether a collective of cognitive architectures (which may be realized as incoherent entities or as elements of a certain meta system) can be used to simulate cognitive behavior has still to be proved.

A. Implementation

This thesis presents an approach for the representation and simulation of human interaction and cognition based on Situation-Operator-Modeling. By means of Situation-Operator-Modeling an intermediate level between the real world and technical models is realized. The step from the real world to a SOM-based description is explained in the previous chapters in detail. In the following, the technical implementation, which corresponds to the step from the SOM-based description to an executable technical model, is focused.

In the context of this thesis, the technical implementation is realized by patterns of high-level Petri Nets and class objects, which are utilized within the Petri Nets and independently as well. The applied Petri Net formalism is called *Reference Nets*, which can be designed and simulated by the software tool *Renew* standing for *Reference Net Workshop*. As a special feature, Reference Nets can contain class objects as tokens which may switch from one place to another if transitions fire. Hence, the formalism Reference Nets belongs to the class of object-oriented Petri Nets. In the case of the software tool *Renew*, class objects are formatted according to the specifications of the higher programming language *Java*.

In addition to class objects and simple standard tokens, the formalism Reference Nets allows to design nets that contain other nets as tokens. Hence, a supernet can contain several subnets. This kind of hierarchization is realized by so-called synchronous channels, allowing to design transitions, e.g., one in a supernet and another one in a corresponding subnet, which fire simultaneously. Thus, one or several tokens may be exchanged between a supernet and a subnet. Besides synchronous channels, the formalism Reference Nets also allows to structure the modeled Petri Nets through fusion places. A fusion place (which is symbolized by a circle with double edging) of a regular place contains the same tokens of that regular place at the same time, but it can be placed to arbitrary position in the same net. Hence, even complex Petri Nets may be visualized in a clear manner.

In the following, the formal representations of situations and operators are illustrated by Petri Nets and UML¹ diagrams.

¹Unified Modeling Language

SOM-based class structure

The implementation is partially realized by SOM-based data structures, which are defined through classes of the programming language Java. After an object of these classes is created, this object is initialized with certain values and it may be related to other objects hierarchically and netlike respectively. In Fig. A.1, the corresponding class structure is shown. Here the relations between the classes

- `Characteristic`,
- `Situation`,
- `Experience`,
- `Assumption`,
- `Goal`,
- `MetaOperator`,
- `ActionModel`, and
- `Plot`

are illustrated. In contrast to that, operators and relations are represented by patterns of high-level Petri Nets. These Petri Net patterns, which contain objects of the listed classes, are detailed in Section A.

The class `Situation` is the central data structure of the whole information processing. In this respect, two lists of the class `Characteristic`, which represent measured and focused characteristics, are the most important attributes. The list representing the measured characteristics is set by a constructor if an object of the class `Situation` is first created. In contrast to that, the focused characteristics, which may correspond to the measured characteristics or represent them in a more abstract manner, are set during information processing (see also Chapter 4). In order to compare two situations, the class `Situation` includes different methods. For example, the method `compare(sit: Situation): Boolean` can be used to estimate whether the related situation corresponds to the situation `sit` taken by the method as attribute. Furthermore, the method `isSubSitOf(sit: Situation): Boolean` may be applied in order to check whether the related situation is a subsituation of the situation `sit` (see also Section 3.1.2). Some of the presented learning algorithms (see Chapter 4.3) require that the characteristics' parameters of a situation may be changed. Therefore, the methods `add(m: String, k: Double)` and `change(m: String, k: Double)`, which are detailed in Section A, can be applied. Finally, also methods for the visualization of situations are available. For example, the method `autoprint()` displays the names and parameters of all focused

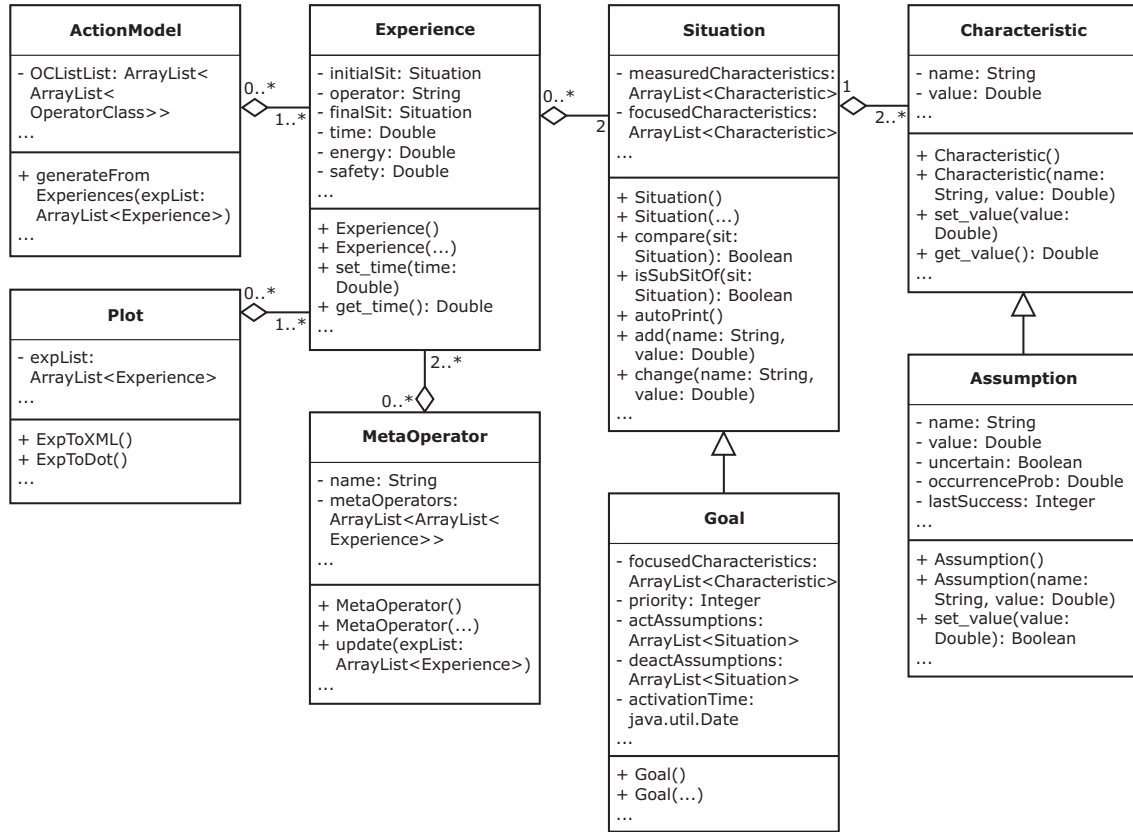


Figure A.1.: Class diagram with SOM-based data structures

characteristics of the related situation in an IDE's² output window.

As already mentioned above, there is an aggregation between the class **Situation** and the class **Characteristic**. An object of the class **Situation** requires at least two objects of the class **Characteristic** in order to describe the measured and focused characteristics as well. If an object of the class **Characteristic** is created, the values of the attributes **name** and **value** may be set by the constructor. However, if the values are to be changed also after an object was created, e.g. by a learning mechanism, the values can also be changed by simple 'set methods', like **set_name(name: String)** or **set_value(value: Double)**. A specialized version of the class **Characteristic** is the class **Assumption**, which also includes the attributes **name** and **value**. Furthermore, the class **Assumption** contains additional attributes as **uncertain**, **occurrenceProb**, and **lastSuccess** (see also 4.1.3), which are utilized by several cognitive functions (see also Chapter 4.3). Lists of the class **Assumption** are applied within operator nets, which is detailed in Section A.

²Integrated Development Environment

In this thesis, situations may also be goals. Hence, the class **Goal** corresponds to a specialization of the class **Situation**. The class **Goal** also contains lists of the class **Characteristic** and includes the same methods as the class **Situation**. Furthermore, the class **Goals** contains additional attributes as **priority**, **actAssumptions**, **deactAssumptions**, and **activationTime** (see also Section 4.2.1).

The key attributes of the class **Experience** are **initSit**, **finalSit** and **operator**, which may be set by the constructor. Hence, there is an aggregation between the class **Situation** and the class **Experience**. Further attributes are **time**, **energy**, and **safety**. The values of these additional attributes can be learned from interaction and may be set by the methods **set_time(time: Double)**, **set_energy(energy: Double)**, and **set_safety(safety: Double)**. Several objects of the class **Experience** can be used to represent a meta operator. Therefore, it is required that the experiences can be arranged in a logical sequence, where the final situation of a certain experience corresponds to the initial situation of a succeeding experience (except for the last element of that sequence). Hence, the related class **MetaOperator** includes an attribute storing a list of lists of the class **Experience**, which can be edited by the constructor and the method **update(expList: ArrayList<Experience>)**.

Action spaces are represented by a set of the class **Experience**. A related list is also utilized by the class **ActionModel**. By the method **generateFromExperiences(expList: ArrayList<Experience>)**, a list of the class **Experience** may be used to derive an action model, which is represented by a list of lists of the class **OperatorClass**. In this regard, a list of the class **OperatorClass** corresponds to a sequence of operator nets (see Section A) and may be processed analogically. Furthermore, several additional methods for the processing of action spaces are implemented. For example, the class **Plot** consists of methods which can be used to convert action spaces into other representations. With the method **ExpToXML()** a set of experiences, which are stored in the attribute **expList**, is converted to a **.xml**-file in order to store the related information persistently. Analogically, the method **ExpToDot()** can be used to visualize action spaces. If the method is executed, a **.dot**-file is created. This **.dot**-file can be interpreted by the software tool *GraphViz* illustrating the action space as directed graph (see also Section A).

Operator nets

In the approach presented in this thesis, operators and relations are represented by special patterns of high-level Petri Nets, which generate and contain the SOM-related class objects explained in Section A. By means of the patterns of high-level Petri Nets, action logic as well as the internal structure of situations may be represented. As a special feature, the net patterns can be adapted and restructured during simulation. Thus, it is possible to model learning abilities of cognitive systems.

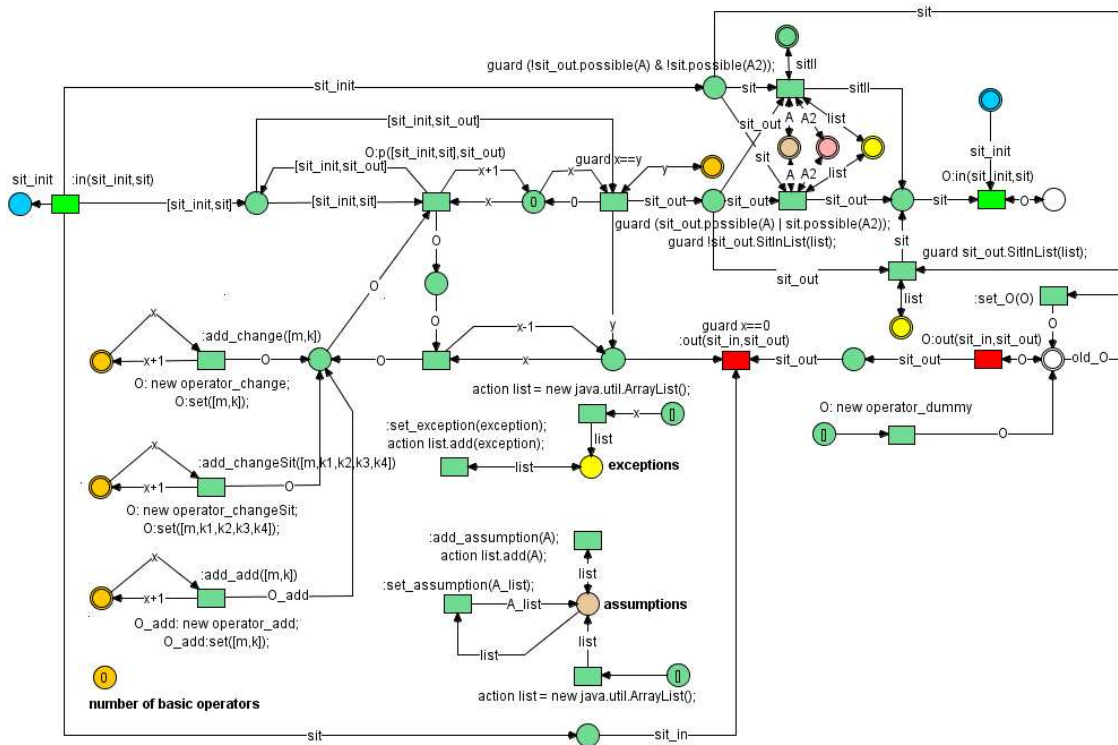


Figure A.2.: Operator net (implemented with Renew)

One of the most important net pattern utilized in this work is illustrated in Fig. A.2. This net pattern is denoted as ‘operator net’ and describes the modification of an input situation depending on certain assumptions (or exceptions). However, in order to represent the function of operators and relations, in general, more than one operator net is required. Hence, several operator nets are interlaced into each other and related as a whole to a certain operator or relation. Accordingly, this kind of net pattern is denoted as a ‘sequence of operator nets’ or ‘operator net sequence’ (see Fig. A.3). Each operator net of an operator net sequence takes an initial situation and a situation that may be modified successively (in the following denoted as situation to be modified) as inputs. Within each single operator net, the assumptions for a modification is checked based on the initial situation. Depending on whether the assumptions are fulfilled or not, the situation to be modified is changed or remains unchanged. After that, the situation to be modified and the initial situation are taken and processed by the next operator net, which repeats for each operator net of a sequence. The last operator net in a sequence contains only a simple dummy net, which forwards the initial situation and the situation to be modified (now the final situation) to the super net containing the sequence of operator nets (see Fig. A.3). In this regard, each operator net initially contains a dummy net. Hence, the dummy net of an operator net sequence may be simply exchanged by a new operator net in order to extend the number of operator nets in that sequence.

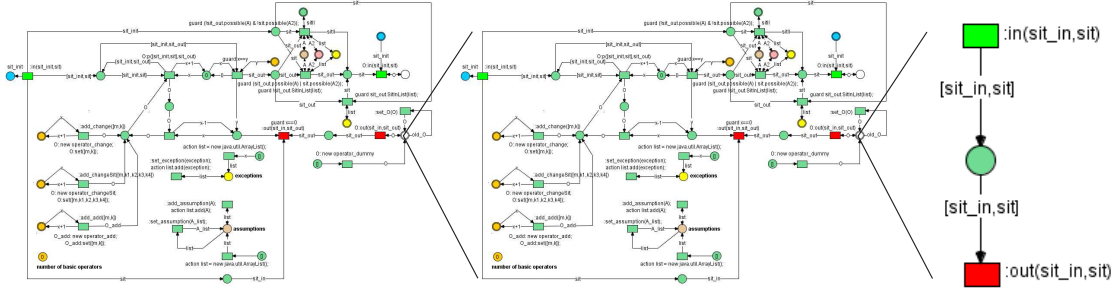


Figure A.3.: Sequence of operator nets (implemented with Renew)

The inputs of operator nets (two objects of the class **Situation**) are taken by means of the synchronous channel `:in(sit_init,sit)`. In the case of the first operator net in a sequence, the values of the variables `sit_init` and `sit`, which are objects of the class **Situation** (taken from the super net) containing the sequence of operator nets, are equal. In all other cases, the values of `sit_init` and `sit` may differ from each other. The variable `sit_init` takes in any case the initial situation, which is used to check whether the assumptions for the modification of a situation are fulfilled. In contrast to that, the variable `sit` takes the situation that is changed successively and that describes the final situation at the end.

Assumptions and exceptions

Assumptions of operators are represented by lists of the class object **Assumption** and are stored within the operator nets. However, in contrast to previous implementations of SOM, here, also exception can be defined. Exceptions are negated assumptions and may also be represented by lists of the class object **Assumption**. The adding and storage of assumptions and exceptions is realized by two separated net patterns within the operator nets. These separated net patterns are placed at the lower right part of the operator nets and are linked with the remaining net via fusion places (see Fig. A.2). Thus, the overall net structure keeps clear, although assumptions and exceptions are processed at different positions. In order to check whether the assumptions and exceptions are fulfilled, the stored lists of the class object **Assumptions** are compared with the initial situation stored in the variable `sit_init`. If the assumptions are fulfilled, the value of the variable `sit_out`, which is the situation to be modified including the changes of the current operator net, is taken by the next operator net in the sequence. Otherwise, if the assumptions are not fulfilled, the value of the variable `sit`, which is the situation to be modified without the changes of the current operator net, is taken by the next operator net in the sequence. This is realized within the upper right part of the operator net including several fusion places that store the lists of the class object **Assumption** (see Fig. A.2).

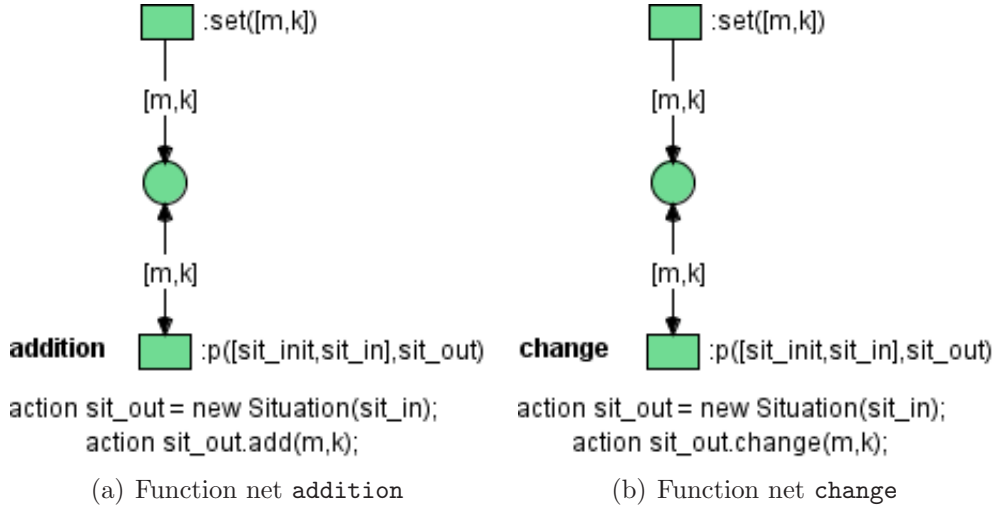


Figure A.4.: Examples for function nets (implemented with Renew)

Function nets

The modification of situations is realized by function nets, which are subnets within the operator nets and added to them via synchronous channels (e.g., `:add_change([m,k])`) located at the bottom left part of an operator net (see Fig. A.2). The final situation of an operator net is generated by applying all stored function nets to the situation to be modified. Whether the resulting situation stored in the variable `sit_out` is also taken by the next operator net in the sequence, depends on the defined assumptions and exceptions, as explained in the previous subsection.

In Fig. A.4, two examples for function nets are shown. Both net patterns consist of a synchronous channel `:set([m,k])`, which is used to define the values of the variables `m` and `k`. The value of the variable `m` is a character string corresponding to the name of a certain characteristic and the value of the variable `k` is numerical. Both net patterns also consist of a second synchronous channel. Thus, a new object of the class `Situation` is created and initialized with the characteristics of the situation to be modified. Furthermore, a method of the created class object, which takes the values of the variables `m` and `k` as attributes, is executed.

Within the function net illustrated in Fig. A.4(a), the method `add(m: String, k: Double)` is executed. Thus, the parameter of the characteristic with the name `m` is increased/decreased by the value `k` (depending on the algebraic sign of `k`). Similarly, the method `change(m: String, k: Double)` is executed within the function net shown in Fig. A.4(b). By the execution of this method, the parameter of the characteristic with the name `m` is exchanged with the value `k`.

Generation of action spaces

The generation of SOM-based action spaces is a key feature of the approach presented in this thesis. These action spaces are inspired by reachability graphs of Petri Nets, but they are not to be confused to them. A reachability graph of a Petri Net contains all states which are reachable related to a certain initial state. In this regard, a state corresponds to a certain configuration of tokens on the Petri Net's places. Furthermore, each edge within the directed graph is related to the firing of a certain transition within the Petri Net. In contrast to that, action spaces, as defined in this thesis, contain situations and operators. Hence, an action space can be used to estimate those sequences of operators that may be performed in order to transfer a certain initial situation into a certain final situation. In this work, however, situations and operators are not related to single places and transitions. As mentioned above, situations are represented by class objects, which are tokens within the Petri Net, and operators are represented by whole patterns of Petri Nets.

In order to generate an action space, all available operators are applied to a certain initial situation and all resulting situations as well. The application of all available operators to an initial situation leads in most cases to several new final situations. In the next step, all available operators are applied again to the final situations resulting from the previous step. This procedure repeats until no more new situations result or if a certain amount of final situations is reached. The result of each calculation is stored within a new object of the class **Experience**, which consists of attributes for the initial situation, the name of the performed operator, and the final situation (see also Section A). Hence, the object of the class **Experiences** represents the function of the performed operators with respect to a special initial situation. Finally, an action space is described by a collection of several objects of the class **Experience**, which may be utilized in order to generate possible paths from an initial situation to a desired situation.

The net pattern in Fig. A.5 is used to generate action spaces according to the description above. Here, all operator net sequences lie on the white-colored place in the upper part of the net. The initial situation and all situations which are created during action space generation are deposited on the gray-colored oval place in the middle. If the oval place contains a situation that was already selected before, the situation is deleted from that place by the transition with the guard function `sit.SitInList(list)`. According to the guard function, which is a method of the class **Situation**, the transition fires only if the related situation (which is selected randomly) is contained in the variable `list`, whose value is a list of objects of the class **Situation**. If a situation on the oval place is not contained in the variable `list`, it may be taken by the transition with the guard function `!sit.SitInList(list)`, which is a negation of the guard function mentioned first. Thus, several variables are updated and the selected situation is added to the variable `list`. After that, the selected situation is deposited on another place in a combined token with the names of the operators to be applied. By the firing of the transition with the synchronous channel `0:in(sit_in, sit_in)`, the selected situation is assigned

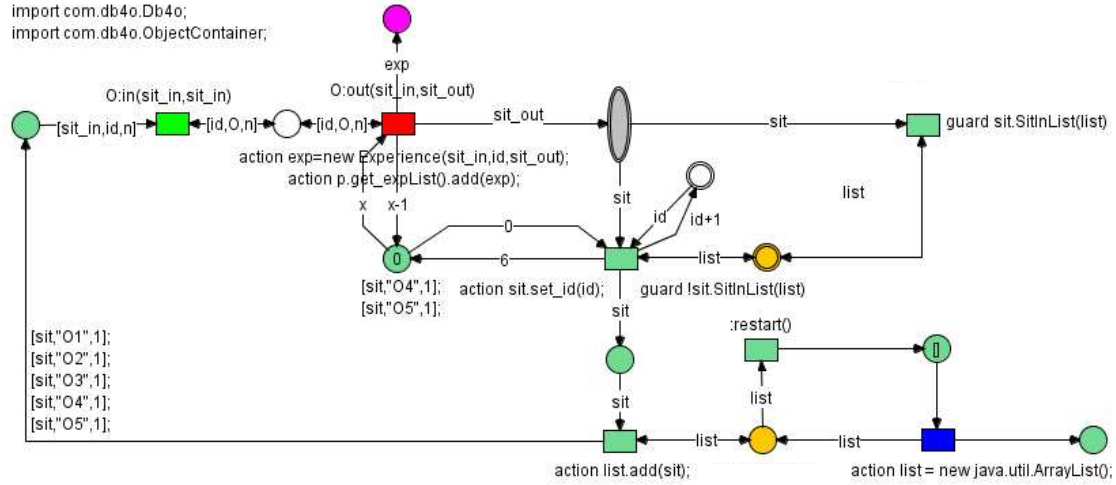


Figure A.5.: Net pattern for the generation of action spaces (implemented with Renew)

to all operator net sequences and by the firing of the transition with the synchronous channel `O:out(sit_in, sit_out)` the final situation is taken back from the operator net sequences. The final situations taken by the variable `sit_out` are deposited on the oval place. Furthermore, an object of the class `Experience` is created. This objects takes the initial situation from the variable `sit_in`, the final situation from the variable `sit_out`, and the name of the related operator and is deposited on a separate place. The sum of all generated objects of the class `Experiences` finally represents the generated action space.

Graphical representation of SOM

The graphical notation of SOM can be used to visualize the interaction within Human-Machine-Systems. In the case of complex systems, the separate visualization of certain parts of the entire model might be helpful. For example, a system's action logic as well as the situation's internal structure can be illustrated by different graphs. These graphs represent the connections between situations and actions or the connections between characteristics and relations respectively. In this regard, a complex model may also be visualized clearly from different perspectives and degrees of abstraction. Hence, this kind of illustration can also be understood easily by inexperienced viewers.

In order to visualize graphs automatically, software tools for graph visualization can be used. A well-known tool is *Graphviz* [GN00, GKN09] providing several programs for graph visualization and layouting. Here, a graph's nodes and edges are described within a `.dot`-file, which can be created with text editors or drawing tools (e.g., *dotty* [KN96]). Another comfortable visualization tool, which can also interpret `.dot`-files, is *ZGRViewer* [Pie05]. The general code and the description of a `.dot`-file for

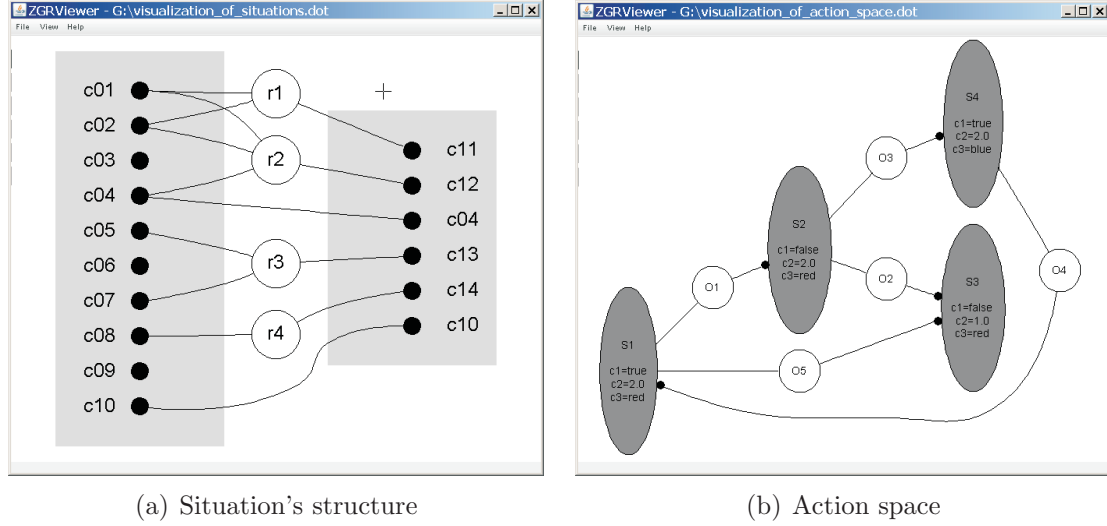


Figure A.6.: SOM-based visualization with Graphviz and ZGRViewer

SOM-related visualization are given in Section A.

A SOM-based model implemented by a high-level Petri Net (see Section A) or another formalisms (see Section 3.2) may be converted to a graph representation and stored as `.dot`-file. If the SOM-based model changes, also the corresponding `.dot`-file can be updated. In addition to that, the software tool ZGRViewer can be triggered (e.g., by hotkey commands) in order to reload the `.dot`-file and to update the visualization. Hence, the relation of particular situations (marked with different colors) and the change of an action space's structure can be observed online. If the model corresponds moreover to a mental representation of the real world, also the observation of a system's internal state (in order to improve the evaluation of its behavior) is possible.

As explained above, the action logic as well as the situation's internal structure can be visualized by graphs. In order to illustrate that, Fig. A.6(a) shows two related examples. In Fig. A.6(a), two situations are shown. The right situation is a derived situation of the left situation. Hence, the right situation contains characteristics which are either taken or derived from the left situation. Accordingly, the lines with white circles represent relations and the lines without white circles represent that the characteristics on both sides are equal. In contrast to the original notation defined in [Söf01c], here, the relations are drawn between both situations in order to focus on the connections of characteristics. An example for an action space is visualized in Fig. A.6(b). In this case, also the names and parameters of characteristics are shown within the situations. Thus, it may be traced how the parameters are changed by the execution of operators.

Persistent storage of SOM-based knowledge

SOM-based knowledge such as certain situations, operators, or whole action spaces has to be stored persistently in order to generate initial knowledge by hand, to analyze or to modify learned knowledge, to save a certain state of knowledge, and to exchange knowledge with other systems or software tools. In this thesis, knowledge is stored persistently within an

- object-oriented database, and
- `.xml`-files.

Both approaches provide different advantages and are used in this work for different kinds of purposes.

With object-oriented databases, it is possible to store and to reload whole object structures (objects containing other objects etc.) Furthermore, object-oriented databases provide generally query languages, which may be used to analyze a contained class structure. In [Ahl07], experiences are also stored in an object-oriented database and the included query mechanism is used to generate plans. In this thesis, the open source object database *db4o* is used to store the proposed action and perception models. This database can easily be included into own applications and provides different kinds of query mechanisms. Moreover, due to the fact that the database and the applied Petri Net formalisms are both based on the programming language Java, an object of the database can be integrated in a Petri Net as token.

The abbreviation XML stands for Extensible Markup Language and it is designed to create hierarchical structured data files, which can be exchanged by different platforms and software. For example, HTML is a specific kind of XML-based file format, which is also denoted as ‘xml-scheme’. Furthermore, XML is provided by a lot of Application Programming Interfaces (APIs) and IDEs. In this thesis, a SOM-based xml-scheme is defined. Those files which are formatted according to that scheme are used to store situations, operators, and action spaces. Furthermore, the SOM-based `.xml`-files may be used to exchange information with other software.

In Listing A.1, an example for a SOM-based xml-files is given. The main element is `ActionSpace`, which contains the elements `Situations` and `Operators`. The element `Situations` contains several elements of the type `Situation`, which further contain one or several elements of the type `Characteristic`. Each element of the type `Situation` has at least one mandatory attribute `id`. This attribute is used to relate the situations to the operators described in the following. Besides the attributes `id`, also further attributes, like the attribute `safety`, can be added in order to store general information about the situations. Each element `characteristic` contains the attributes `name` and `value`.

Operators are described by the element **Operator** contained in the element **Operators**. Each element **Operator** contains the attributes **name**, **initSit**, and **finalSit**. The attribute **name** stores the name of the operator and the attributes **initSit** and **finalSit** store the ids of the situations defined in the element **Situations**. In the example (see Listing A.1), the operator o_1 transfers the situation s_1 to the situation s_2 . The operator o_2 transfers the situation s_1 as well as the situation s_2 to the situation s_3 . Besides the three attributes contained in the example, also further attributes, like **energy**, **safety**, and **time**, may be considered. However, if only information about situations are to be stored or exchanged, the element **ActionSpace** and the element **Operators** can be neglected.

Listing A.1: Example for a SOM-based .xml-file

```
<ActionSpace>
  <Situations>
    <Situation id="1" safety="1">
      <Characteristic name="c1" value="0.0"></Characteristic>
      <Characteristic name="c2" value="30.0"></Characteristic>
      <Characteristic name="c3" value="2.0"></Characteristic>
    </Situation>
    <Situation id="2" safety="0.8">
      <Characteristic name="c1" value="1.0"></Characteristic>
      <Characteristic name="c2" value="30.0"></Characteristic>
      <Characteristic name="c3" value="2.0"></Characteristic>
    </Situation>
    <Situation id="3" safety="1">
      <Characteristic name="c1" value="1.0"></Characteristic>
      <Characteristic name="c2" value="5.0"></Characteristic>
      <Characteristic name="c3" value="2.0"></Characteristic>
    </Situation>
  </Situations>
  <Operators>
    <Operator name="O1" initSit="1" finalSit="2"></Operator>
    <Operator name="O2" initSit="1" finalSit="3"></Operator>
    <Operator name="O2" initSit="2" finalSit="3"></Operator>
  </Operators>
</ActionSpace>
```

Index

- ACT-R, 14, 25, 31
- Action model, 65
- Action space, 46, 65
- ADAPT, 26
- Agent, 10
- Allocation of Resources (AoR), 24
- Artificial Intelligence (AI), 10
- Artificial Neural Network (ANN), 18
- Assumption, *see* SOM
- Attention model, 69
- Attention module, 74
- Autonomous system, 1
- Autonomy, 1

- Backpropagation, 19
- Backward chaining, 14
- Basic operator, 99

- Case-based Reasoning, 20
- Cerebus, 27
- Characteristic, *see* SOM
- Chunk, 31
- Chunking, 20
- CLARION, 27
- CogAff scheme, 26
- Cognition, 7
- Cognitive Architecture, 21
- Cognitive Psychology, 9
- Cognitive Robotics, 7
- Cognitive Technical Systems, 1, 7
- Cognitivist approach, 22
- CoLiDeS, 27
- Coloured Petri Nets, 49
- Combined Intelligent System, 12
- Conflict, 63, 64
- Connectionist approach, 12, 18
- Controller program, 99
- Correlation, 20

- Coupled Intelligent System, 12

- Data Mining, 16
- Decision-Tree-Learning, 17
- Derived characteristic, 58
- Derived situation, 45
- Dynamic Programming, 20

- EM-ONE, 25
- Emergent systems approach, 22, 25
- EPIC, 14, 25
- Execution, 24
- Experience, 59, 60

- Filter program, 99
- First-order logic, 13
- Focused characteristic, 58
- Forward chaining, 14
- Frames, 15
- Fuzzy logic, 13

- General Problem Solver (GPS), 24
- Genetic Algorithms, 20
- Goal, 70
- Grounded Situation Model (GSM), 33

- HCogAff, 26
- High-level Petri Net (HPN), 48
- Human brain, 8
- Human error, 42, 51
- Human Model of Reference, 23
- Hybrid approach, 22, 27
- Hybrid Intelligent System, 12

- ICARUS, 26
- ILCA architecture, 89
- Integrated theory of mind, 21
- Intelligent Machine Architecture, 30
- Interpretation, 23

-
- ISAC, 30
 - Knowledge Representation, 11
 - Knowledge-based behavior, 23
 - Learning, 1
 - LiCAI, 27
 - LIDA framework, 28
 - Logic Theorist, 24
 - Machine Learning, 16
 - Macro-cognition, 22
 - Markov Decision Process (MDP), 20
 - Measured characteristic, 58
 - Memory/Knowledge Base (KB), 24
 - Mental action space, 91
 - Mental model, 11
 - Meta action space, 46
 - Meta operator, *see* SOM, 67
 - Micro-cognition, 22
 - microPsi, 26
 - Middleware, 98
 - Neuroscience, 8, 9
 - Ontology, 12
 - Operator, *see* SOM
 - Operator net, 61
 - Parameter, *see* SOM
 - Pattern Recognition, 16
 - Perception, 23
 - Perception model, 69
 - Pioneer 3 DX, 100
 - PIPE, 23
 - Planning, 14, 23
 - Polyscheme framework, 27
 - Prodigy, 26
 - Production rule, 31
 - Production system, 14
 - Propositional logic, 13
 - Psi theory, 26
 - Q-learning, 20
 - Reasoning, 12
 - Recognition model, 69
 - Recognition module, 73
 - Reference Nets, 49
 - Regression Analysis, 20
 - Reinforcement learning, 17, 19
 - Relation, *see* SOM
 - Rigidity, 43
 - Ripley, 33
 - Robonaut, 32
 - Rule-based behavior, 23
 - Search algorithm, 15
 - Semantic Network (SN), 15
 - Semantic Web, 13
 - Sensory EgoSphere, 30
 - Short-term memory, 79
 - Situation, *see* SOM
 - Situation Calculus, 14
 - Situation-Operator-Modeling, *see* SOM
 - Skill-based behavior, 23
 - Soar, 14, 25
 - Society of Mind, 25
 - SOM, 34, 41
 - Assumption, 62
 - Characteristic, 41
 - Meta operator, 41
 - Operator, 41
 - Parameter, 41
 - Relation, 41
 - Situation, 41, 57, 58
 - State space analysis, 50
 - State space representation, 15
 - Step ladder model, 22
 - Subsituation, 44
 - Subsumption Architecture, 25
 - Subsymbolic system, 12
 - Supervised learning, 17
 - Support-Vector-Machines, 19
 - Symbolic system, 12
 - Temporal logic, 13
 - Transformational Intelligent System, 12
 - Uncertainty, 63
 - Unsupervised learning, 17

Literature

- [ABB⁺04] Anderson, J. R.; Bothell, D.; Byrne, M. D.; Douglass, S.; Lebiere, C.; Qin, Y.: An Integrated Theory of the Mind. In: *Psychological Review*, volume 111(4):pp. 1036–1060, 2004.
- [Ahl07] Ahle, E.: *Autonomous Systems: A Cognitive-Oriented Approach Applied to Mobile Robotics*. Dr.-Ing. thesis, Faculty of Engineering Sciences, University of Duisburg-Essen, Duisburg, Juli 2007. Published at Shaker Verlag, Aachen, 2007.
- [AL98] Anderson, J. R.; Lebiere, C.: *The Atomic Components of Thought*. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1998.
- [Alp00] Alpar, P.: *Data Mining im praktischen Einsatz: Verfahren und Anwendungsfälle für Marketing, Vertrieb, Controlling und Kundenunterstützung*. Vieweg, Braunschweig/Wiesbaden, 2000.
- [Alp10] Alpaydin, E.: *Introduction to Machine Learning*. MIT Press, Cambridge, MA, USA, 2nd edition, 2010.
- [And07] Anderson, J. R.: *Kognitive Psychologie*. Springer-Verlag, Berlin/Heidelberg, 6th edition, 2007.
- [Ari04] Aristotele: *Aristotle Metaphysics*. Penguin Books, London, England, 2004. (translation and introduction by H. Lawson-Tancred).
- [Art10] Artsoft Entertainment - Rocks'n'Diamonds. online, last updated June 15th, 2010. URL <http://www.artsoft.org/rocksndiamonds/> (retrieved on December 2nd, 2010).
- [AS08] Ahle, E.; Söffker, D.: Interaction of Intelligent and Autonomous Systems - Part II: Realization of Cognitive Technical Systems. In: *MCMDs - Mathematical and Computer Modelling of Dynamical Systems*, volume 14(4):pp. 319–339, 2008.
- [AW94] Åström, K. J.; Wittenmark, B.: *Adaptive Control*. Addison-Wesley, Reading, MA, USA, 2nd edition, 1994.
- [AWWB09] Affenzeller, M.; Winkler, S.; Wagner, S.; Beham, A.: *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Chapman & Hall, Boca Raton, FL, USA, 2nd edition, 2009.

-
- [Bac05] Bach, J.: MiniPsi - Der Mac-Roboter. In: *metamac magazin*, volume 46:pp. 11–14, 2005.
 - [Bac09] Bach, J.: *Principles of Synthetic Intelligence - PSI: An Architecture of Motivated Cognition*. Oxford University Press, New York, NY, USA, 2009.
 - [Bau96] Baumgarten, B.: *Petri Netze - Grundlagen und Anwendungen*. Spektrum Akademischer-Verlag, Heidelberg/Berlin/Oxford, 2nd edition, 1996.
 - [BBV06] Bach, J.; Bauer, C.; Vuine, R.: MicroPsi: Contributions to a Broad Architecture of Cognition. In: *KI 2006: Advances in Artificial Intelligence*. Bremen, Germany, 2006.
 - [Bec03] Beckstein, C.: *Handbuch der Künstlichen Intelligenz*, chapter Suche, pp. 125–151. Oldenbourg Wissenschaftsverlag, Munich, Germany, 4th edition, 2003.
 - [Bel57a] Bellman, R. E.: A Markovian Decision Process. In: *Journal of Mathematics and Mechanics*, volume 6, 1957.
 - [Bel57b] Bellman, R. E.: *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.
 - [BFOS84] Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J.: *Classification and Regression Trees*. Chapman & Hall, New York, NY, USA, 1984.
 - [Bis06] Bishop, C. M.: *Pattern Recognition and Machine Learning*. Springer-Verlag, New York, NY, USA, 2006.
 - [BL04] Brachman, R. J.; Levesque, H. J.: *Knowledge Representation and Reasoning*. Morgan Kaufmann, San Francisco, CA, USA, 2004.
 - [BL06] Best, B. J.; Lebiere, C.: *Cognition and Multi-Agent Interaction: From Cognitive Modeling to Social Simulation*, chapter Cognitive agents interacting in real and virtual worlds, pp. 186–218. Cambridge University Press, New York, NY, USA, 2006.
 - [BLHL01] Berners-Lee, T.; Hendler, J.; Lassila, O.: The Semantic Web: a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. In: *Scientific American*, volume 284(5):pp. 34–43, 2001.
 - [BLL04] Benjamin, D. P.; Lyons, D.; Lonsdale, D.: ADAPT: A cognitive architecture for robotics. In: *Proc. of the Int. Conf. of Cognitive Modeling*. 2004.
 - [BMH⁺09] Ball, J.; Myers, C.; Heiberg, A.; Cooke, N.; Matessa, M.; Freiman, M.: The Synthetic Teammate Project. In: *Proc. of the 18th Conf. on Behavior Representation in Modeling and Simulation*. 2009.

- [BMS⁺05] Burghart, C. R.; Mikut, R.; Stiefelhagen, R.; Asfour, T.; Holzapfel, H.; Steinhaus, P.; Dillmann, R.: A cognitive architecture for a humanoid robot: A first approach. In: *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, pp. 357–362. 2005.
- [Bob01] Bobko, P.: *Correlation and Regression: Applications for Industrial Organizational Psychology and Management*. Sage Publications, Thousand Oaks/London/New Delhi, 2nd edition, 2001.
- [Bra02] Brachman, R.: Systems that know what they’re doing. In: *IEEE Intelligent Systems*, volume 17(6):pp. 67–71, 2002.
- [Bre01] Breazeal, C.: Socially intelligent robots: research, development, and applications. In: *2001 IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pp. 2121–2126. Tucson, AZ, USA, 2001.
- [Bro91] Brooks, R. A.: Intelligence without representation. In: *Artificial Intelligence*, volume 47:pp. 139–159, 1991.
- [Bro99] Brooks, R. A.: *Cambrian Intelligence: The Early History of the New AI*. MIT Press, Boston, MA, USA, 1999.
- [Bur07] Burghart, C. R.: *Mensch und Maschine - Interaktion mit intelligenten Robotersystemen*. Habilitation thesis, Fakultät für Informatik, Universität Karlsruhe, 2007.
- [Cac98] Cacciabue, P. C.: *Modelling and Simulation of Human Behaviour in System Control, Advances in Industrial Control*. Springer-Verlag, London, 1998.
- [Cas02] Cassimatis, N. L.: *Polyscheme: A Cognitive Architecture for Integrating Multiple Representation and Inference Schemes*. PhD thesis, Media Laboratory, Massachusetts Institute of Technology, 2002.
- [Cas03] Cassimatis, N. L.: A Framework for Answering Queries using Multiple Representation and Inference Techniques. In: *Proc. of the 10th Int. Workshop on Knowledge Representation meets Databases*. 2003.
- [CBB⁺10] Cassimatis, N.; Bignoli, P.; Bugajska, M.; Dugas, S.; Kurup, U.; Murugesan, A.; Bello, P.: An Architecture for Adaptive Algorithmic Hybrids. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, volume 40(3):pp. 903–914, 2010.
- [CEG⁺90] Carbonell, J. G.; Etzioni, O.; Gil, Y.; Joseph, R.; Knoblock, C. A.; Minton, S.; Veloso, M. M.: PRODIGY: An Integrated Architecture for Planning and Learning. In: *SIGART Bulletin*, volume 2(4):pp. 51–55, 1990.

-
- [CH00] Christensen, W. D.; Hooker, C. A.: An Interactivist-Constructivist Approach to Intelligence: Self-Directed Anticipative Learning. In: *Philosophical Psychology*, volume 13(1):pp. 5–45, 2000.
 - [CN10] Ching, W.-K.; Ng, M. K.: *Markov Chains: Models, Algorithms and Applications*. Springer, New York, NY, USA, 2010.
 - [CST00] Cristianini, N.; Shawe-Taylor, J.: *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, MA, USA, 2000.
 - [CTBS04] Cassimatis, N. L.; Trafton, J. G.; Bugajska, M.; Schultz, A. C.: Integrating Cognition, Perception, and Action through Mental Simulation in Robots. In: *Robotics and Autonomous Systems*, volume 49:pp. 13–23, 2004.
 - [CTN07] Chong, H.-Q.; Tan, A.-H.; Ng, G.-W.: Integrated cognitive architectures: a survey. In: *Artificial Intelligence Review*, volume 28(2):pp. 103–130, 2007.
 - [Di’78] Di’Simoni, F.: *Token Test for Children: A Manual*. DLM Teaching Resources, Beltsville, MD, USA, 1978.
 - [Dij59] Dijkstra, E. W.: A note on two problems in connexion with graphs. In: *Numerische Mathematik*, volume 1:pp. 269–271, 1959.
 - [DNBH07] Donoghue, J. P.; Nurmikko, A.; Black, M.; Hochberg, L. R.: Assistive technology and robotic control using motor cortex ensemble-based neural interface systems in humans with tetraplegia. In: *The Journal of Physiology*, volume 579:pp. 603–611, 2007.
 - [Dob00] Dobelle, W. H.: Artificial Vision for the Blind by Connecting a Television Camera to the Visual Cortex. In: *ASAIO Journal*, volume 46:pp. 3–9, 2000.
 - [Dör97] Dörner, D.: *The Logic of Failure: Recognizing and Avoiding Error in Complex Situations*. Perseus Publishing, Cambridge, MA, USA, 1997.
 - [Dör01] Dörner, D.: *Bauplan für eine Seele*. rororo, Reinbek, Germany, 2nd edition, 2001.
 - [DSD01] Dörner, D.; Schaub, H.; Detje, F.: Das Leben von Ψ - Über das Zusammenspiel von Kognition, Emotion und Motivation - oder: Eine einfache Theorie komplexer Verhaltensweisen. In: *Sozionik aktuell*, volume 2, 2001.
 - [EFK⁺08] Erdemir, E.; Frankel, C. B.; Kawamura, K.; Gordon, S. M.; Thornton, S.; Ulutas, B.: Towards a cognitive robot that uses internal rehearsal to learn affordance relations. In: *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2016–2021. 2008.

- [EGVS10] Ertle, P.; Gamrad, D.; Voos, H.; Söffker, D.: Action Planning for Autonomous Systems with respect to Safety Aspects. In: *2010 IEEE Int. Conf. on Systems, Man, and Cybernetics*, pp. 2465–2472. Istanbul, Turkey, 2010.
- [EHHP06] Edlich, S.; Hörning, R.; Hörning, H.; Paterson, J.: *The Definitive Guide to db4o*. Apress, Berkeley, CA, USA, 2006.
- [FGM⁺09] Fu, X.; Gamrad, D.; Mosebach, H.; Lemmer, K.; Söffker, D.: Modelling and Implementation of Cognitive-based Supervision and Assistance. In: *MATHMOD 2009 - 6th Vienna Int. Conf. on Mathematical Modelling*, pp. 2063–2068. Vienna, Austria, 2009.
- [FKHB06] Fong, T.; Kunz, C.; Hiatt, L. M.; Bugajska, M.: The human-robot interaction operating system. In: *Proc. of the 1st ACM SIGCHI/SIGART Conf. on Human-Robot-Interaction*. 2006.
- [FP06] Franklin, S.; Patterson, F. G.: The LIDA Architecture: Adding New Modes of Learning to an Intelligent, Autonomous, Software Agent. In: *Proc. of Integrated Design and Process Technology*. 2006.
- [Fra03] Franklin, S.: IDA: A Conscious Artifact? In: *Journal of Consciousness Studies*, volume 10:pp. 47–66, 2003.
- [Ful06] Fulcher, J., editor: *Advances in Applied Artificial Intelligence*. Idea Group Publishing, Hershey, PA, USA, 2006.
- [Gam06] Gamrad, D.: *Development of high-level Petri Net Patterns for computer-aided Simulation and Analysis of Situation-Operator-Models (In German)*. Diploma thesis, Chair of Dynamics and Control, University of Duisburg-Essen, Duisburg, 2006.
- [Gar83] Gardner, H.: *Frames of mind: The theory of multiple intelligences*. Basic Books, New York, NY, USA, 1983.
- [GBG⁺05] Gluck, K. A.; Ball, J. T.; Gunzelmann, G.; Krusmark, M. A.; Lyon, D. R.; Cooke, N. J.: A prospective look at a synthetic teammate for UAV applications. In: *Proc. of the American Institute of Aeronautics and Astronautics Infotech@Aerospace Conference*. 2005.
- [GGNZ06] Guyon, I.; Gunn, S.; Nikravesh, M.; Zadeh, L. A., editors: *Feature Extraction: Foundations and Applications*. Springer-Verlag, Berlin/Heidelberg, 2006.
- [GH06] Gordon, S. M.; Hall, J.: System integration with working memory management for robotic behavior learning. In: *Proc. of the 5th Int. Conf. on Development and Learning*. 2006.

- [GKN09] Gansner, E. R.; Koutsofios, E.; North, S. C.: Drawing graphs with dot. technical report, AT&T/Bell Laboratories, Murray Hill, NJ, USA, December 2009.
- [GN00] Gansner, E. R.; North, S. C.: An open graph visualization system and its applications to software engineering. In: *Software - Practice and Experience*, volume 30(11):pp. 1203–1233, 2000.
- [GNT04] Ghallab, M.; Nau, D.; Traverso, P.: *Automated Planning - Theory and Practice*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.
- [Gol89] Goldberg, D. E.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Bonn, Germany, 1989.
- [GOS07] Gamrad, D.; Oberheid, H.; Söffker, D.: Supervision of Open Systems using a Situation-Operator-Modeling Approach and Higher Petri Net Formalisms. In: *Proc. of 2007 IEEE Int. Conf. on Systems, Man, and Cybernetics*, pp. 925–930. Montréal, Canada, 2007.
- [GOS08] Gamrad, D.; Oberheid, H.; Söffker, D.: Formalization and Automated Detection of Human Errors. In: *SICE Int. Conf. on Instrumentation, Control and Information Technology*, pp. 1761–1766. Tokia, Japan, 2008.
- [GOS09] Gamrad, D.; Oberheid, H.; Söffker, D.: Automated Detection of Human Errors based on Multiple Partial State Spaces. In: *MATHMOD 2009 - 6th Vienna International Conference on Mathematical Modelling*, pp. 651–659. Vienna, Austria, 2009.
- [GP06] Goertzel, B.; Pennachin, C., editors: *Artificial General Intelligence*. Springer-Verlag, Berlin/Heidelberg, 2006.
- [Gra18] Gray, H.: *Anatomy of the human body*. Lea & Febiger, Philadelphia, PA, U'SA, 20th edition, 1918. Bartleby.com, 2000, <http://www.bartleby.com/107/189/>, April 9th, 2010.
- [GRS03] Görz, G.; Rollinger, C.-R.; Schneeberger, J., editors: *Handbuch der Künstlichen Intelligenz*. Oldenbourg Wissenschaftsverlag, Munich, Germany, 4th edition, 2003.
- [Gru93] Gruber, T. R.: A Translation Approach to Portable Ontology Specifications. In: *Knowledge Acquisition*, volume 5(2):pp. 199 – 220, 1993.
- [GS07] Gamrad, D.; Söffker, D.: Konzeption und Implementierung eines kognitiven Fahrer-Fahrzeugüberwachungs- und assistenzsystems. In: *Fahrer im 21. Jahrhundert - Human Machine Interface*, pp. 285–290. Brunswick, Germany, 2007.

-
- [GS09] Gamrad, D.; Söffker, D.: Architecture for Cognitive Technical Systems allowing Learning from Interaction with Unknown Environments. In: *7th Workshop on Advanced Control and Diagnosis (ACD 2009)*. Zielona Góra, Poland, 2009.
- [GS10a] Gamrad, D.; Söffker, D.: Learning from conflicts in real world environments for the Realization of Cognitive Technical Systems. In: *2010 IEEE Int. Conf. on Systems, Man, and Cybernetics*, pp. 1995–2002. Istanbul, Turkey, 2010.
- [GS10b] Gamrad, D.; Söffker, D.: Representation of Knowledge for Technical Systems to Realize Cognitive Functions for Mobile Robotic Applications. In: *Entwurf komplexer Automatisierungssysteme - EKA 2010*, pp. 273–282. Magdeburg, Germany, 2010.
- [Hac05] Hacker, W.: *Allgemeine Arbeitspsychologie: Psychische Regulation von Wissens-, Denk- und körperlicher Arbeit*. Verlag Hans Huber, Berne, Switzerland, 2nd edition, 2005.
- [Has08] Hasselberg, A.: *Extension of a Petri-Net-based Simulation Environment for the Automated Detection of Human Errors (In German)*. Studienarbeit, Chair of Dynamics and Control, University of Duisburg-Essen, Duisburg, 2008.
- [Hel06] Helbig, H.: *Knowledge Representation and the Semantics of Natural Language*. Springer-Verlag, Berlin/Heidelberg, 2006.
- [HFH⁺09] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I. H.: The WEKA Data Mining Software: An Update. In: *SIGKDD Explorations*, volume 11(1), 2009.
- [HK06] Han, J.; Kamber, M.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2nd edition, 2006.
- [HNR68] Hart, P. E.; Nilsson, N. J.; Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In: *IEEE Transactions on Systems Science and Cybernetics*, volume 4(2):pp. 100–107, 1968.
- [Hor01] Horswill, I.: Tagged behavior based systems: integrating cognition with embodied activity. In: *IEEE Intelligent Systems*, volume 16(5):pp. 30–37, 2001.
- [HR83] Hayes-Roth, F.: *Machine Learning: An Artificial Intelligence Approach*, chapter Using Proofs and Refutation to Learn from Experience, pp. 221–240. Morgan Kaufmann Publishers, Los Altos, CA, USA, 1983.
- [HS10] Henning, M.; Spruiell, M.: *Distributed Programming with Ice*. ZeroC, Inc., Palm Beach Gardens, FL, USA, revision 3.4-it1 edition, September 2010.

- [Hul52] Hull, C. L.: *A Behavior System - An Introduction to Behavior Theory Concerning the Individual Organism*. Yale University Press, New Haven, CT, USA, 1952.
- [HV96] Haigh, K.; Veloso, M.: Interleaving planning and robot execution for asynchronous user requests. In: *Proc. of the Int. Conf. on Intelligent Robots and Systems*, pp. 148–155. Osaka, Japan, 1996.
- [JCK06] Jensen, K.; Christensen, S.; Kristensen, L. M.: *CPN Tools State Space Manual*. Univ. of Aarhus, Dep. of Computer Science, Aarhus, 2006.
- [Jen97] Jensen, K.: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1-3. Springer-Verlag, Berlin/Heidelberg/New York, 1997.
- [JKW07] Jensen, K.; Kristensen, L. Michael; Wells, L.: Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. In: *International Journal on Software Tools for Technology Transfer*, volume 9:p. 213–254, 2007.
- [JL83] Johnson-Laird, P. N.: *Mental Models*. Cambridge University Press, Cambridge, England, 1983.
- [JM03] Jenkins, O. C.; Matarić, M. J.: Automated Derivation of Behavior Vocabularies for Autonomous Humanoid Motion. In: *Proc. 2nd Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pp. 225–232. Melbourne, Australia, 2003.
- [Jon06] Jones, J. L.: Robots at the tipping point: the road to iRobot Roomba. In: *IEEE Robotics & Automation Magazine*, volume 13(1):pp. 76–78, 2006.
- [KBP00] Kitajima, M.; Blackmon, M. H.; Polson, P. G.: A comprehension-based model of web navigation and its application to web usability analysis. In: McDonald, S.; Waern, Y.; Cockton, G., editors, *People and Computers XIV - Usability or Else! (Proceedings of HCI 2000)*, pp. 357–373. Springer, New York, 2000.
- [KE08] Krichmar, J. L.; Edelman, G. M.: Design Principles and Constraints Underlying the Construction of Brain-Based Devices. In: *Lecture Notes in Computer Science, Neural Information Processing*, volume 4985:pp. 157–166, 2008.
- [KM97] Kieras, D.; Meyer, D. E.: An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. In: *Human-Computer Interaction*, volume 12:pp. 391–438, 1997.
- [KN96] Koutsofios, E.; North, S. C.: Editing graphs with dotty. technical report 96b (06-24-96), AT&T/Bell Laboratories, Murray Hill, NJ, USA, June 1996.

-
- [KP97a] Kitajima, M.; Polson, P. G.: A comprehension-based model of exploration. In: *Human-Computer Interaction*, volume 12(4):pp. 345–389, 1997.
- [KP97b] Kitajima, M.; Polson, P. G.: LICA+: A Comprehension-Based Model of Learning for Display-Based Human-Computer Interaction. In: *ACM conference on human factors in computing systems*, pp. 333–334. 1997.
- [KPB⁺04] Kawamura, K.; Peters II, R. A.; Bodenheimer, R. E.; Sarkar, N.; Park, J.; Clifton, C. A.; Spratley, A. W.: A Parallel Distributed Cognitive Control System for a Humanoid Robot. In: *Int. Journal of Humanoid Robotics*, volume 1(1):pp. 65–93, 2004.
- [Kum02] Kummer, O.: *Referenznetze*. Logos Wissenschaftsverlag, Berlin, Germany, 2002.
- [Kur05] Kurzweil, R.: *The Singularity Is Near: When Humans Transcend Biology*. Viking Press, New York, NY, USA, 2005.
- [KWD08] Kummer, O.; Wienberg, F.; Duvigneau, M.: *Renew – User Guide*. University of Hamburg, Department for Informatics, Theoretical Foundations Group, Distributed Systems Group, Hamburg, 2nd edition, July 2008.
- [Lan06] Langley, P.: Cognitive Architectures and General Intelligent Systems. In: *AI Magazine*, volume 27(2):pp. 33–44, 2006.
- [LC04] Langley, P.; Cummings, K.: Hierarchical skills and cognitive architectures. In: *Proc. of the Twenty-Sixth Annual Conf. of the Cognitive Science Society*, pp. 779–784. 2004.
- [LC06] Langley, P.; Choi, D.: A unified cognitive architecture for physical agents. In: *Proc. of the Twenty-First National Conf. on Artificial Intelligence*. AAAI Press, Boston, 2006.
- [LCR09] Langley, P.; Choi, D.; Rogers, S.: Acquisition of hierarchical reactive skills in a unified cognitive architecture. In: *Cognitive Systems Research*, volume 10:pp. 316–332, 2009.
- [Lea96] Leake, D. B., editor: *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press, Menlo Park, CA, USA, 1996.
- [LLR08] Langley, P.; Laird, J. E.; Rogers, S.: Cognitive architectures: Research issues and challenges. In: *Cognitive Systems Research*, volume 10(2):pp. 141–160, June 2008.
- [LNR87] Laird, J. E.; Newell, A.; Rosenbloom, P. S.: Soar: An architecture for general intelligence. In: *Artificial Intelligence*, volume 33(1):pp. 1–64, 1987.

- [LRN86] Laird, J. E.; Rosenbloom, P. S.; Newell, A.: Chunking in Soar: The Anatomy of a General Learning Mechanism. In: *Machine Learning*, volume 1:pp. 11–46, 1986.
- [Lug01] Luger, G. F.: *Künstliche Intelligenz: Strategien zur Lösung komplexer Probleme*. Pearson Studium, Munich, Germany, 4th edition, 2001.
- [Mac98] MacKintosh, N. J.: *IQ and Human Intelligence*. Oxford University Press, New York, NY, USA, 1998.
- [Mav07] Mavridis, N.: *Grounded Situation Models for Situated Conversational Assistants*. PhD thesis, Media Laboratory, Massachusetts Institute of Technology, 2007.
- [McC63] McCarthy, J.: Situations, actions, and causal laws. Stanford Artificial Intelligence project, Memo No. 2, Stanford University, Stanford, CA, USA, July 3rd, 1963.
- [McC07] McCarthy, J.: What is artificial intelligence? online, last updated November 12th, 2007. URL <http://www-formal.stanford.edu/jmc/whatisai/whatisai.html> (retrieved on December 2nd, 2010).
- [Mic86] Michalski, R. S.: *Machine Learning: An Artificial Intelligence Approach*, volume 2, chapter Understanding the Nature of Learning: Issues and Research Directions, pp. 3–25. Morgan Kaufmann Publishers, Los Altos, CA, USA, 1986.
- [Min74] Minsky, M.: A Framework for Representing Knowledge. technical report Memo 306, MIT, AI Laboratory, Cambridge, MA, USA, June 1974.
- [Min86] Minsky, M.: *The Society of Mind*. Simon & Schuster, New York, NY, USA, 1986.
- [Min91] Minsky, M.: Logical Versus Analogical or Symbolic Versus Connectionist or Neat Versus Scruffy. In: *AI Magazine*, volume 12(2):pp. 34–51, 1991.
- [Min06] Minsky, M.: *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. Simon & Schuster, New York, NY, USA, 2006.
- [Mob04] MobileRobots Inc, Amherst, NH, USA: *PTZ Robotic Cameras*, Version 4, October 2004.
- [Mob07] MobileRobots Inc, Amherst, NH, USA: *Pioneer Gripper Manual*, Version 7.1, January 2007.

-
- [Mob08] MobileRobots Inc, Amherst, NH, USA: *ACTS User Manual*, Version 6.5, November 2008.
 - [Mob09] MobileRobots Inc, Amherst, NH, USA: *ARNL Installation & Operations Manual*, Version 4, August 2009.
 - [Mob10] MobileRobots Inc, Amherst, NH, USA: *Pioneer 3 Operations Manual*, Version 6, September 2010.
 - [MOW08] Möhlenbrink, C.; Oberheid, H.; Werther, B.: *Human Factors for assistance and automation*, chapter A model based approach to Cognitive Work Analysis and work process design in air traffic control, pp. 401–414. Shaker Publishing, Maastricht, the Netherlands, 2008.
 - [MR06] Mavridis, N.; Roy, D.: Grounded Situation Models for Robots: Where words and percepts meet. In: *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2006.
 - [MWK⁺06] Mierswa, I.; Wurst, M.; Klinkenberg, R.; Scholz, M.; Euler, T.: YALE: Rapid Prototyping for Complex Data Mining Tasks. In: *Proc. of the 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD-06)*. 2006.
 - [NBFK03] Nauck, D.; Borgelt, C.; F.Klawonn; Kruse, R.: *Neuro-Fuzzy-Systeme: Von den Grundlagen künstlicher Neuroner Netze zur Kopplung mit Fuzzy-Systemen*. Vieweg, Wiesbaden, Germany, 3rd edition, 2003.
 - [New90] Newell, A.: *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA, USA, 1990.
 - [NRPH00] Norgaard, M.; Ravn, O.; Poulsen, N. K.; Hansen, L. K.: *Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook*. Springer-Verlag, London/Berlin/Heidelberg, 2000.
 - [NS56] Newell, A.; Simon, H. A.: The logic theory machine - A complex information processing system. In: *IRE Transactions on Information Theory*, volume 2(3):pp. 61–79, 1956.
 - [NS61] Newell, A.; Simon, H. A.: GPS: A program that simulates human thought. In: Billings, H., editor, *Lernende Automaten*, pp. 109–124. R. Oldenbourg Verlag, Munich, Germany, 1961.
 - [Pet62] Petri, C. A.: *Kommunikation mit Automaten*. Dr.-Ing. thesis, Schriften des Institutes für Instrumentelle Mathematik, Universität Bonn, Bonn, 1962.
 - [PHKW01] Peters II, R. A.; Hambuchen, K. A.; Kawamura, K.; Wilkes, D. M.: The sensory egosphere as a short-term memory for humanoids. In: *Proc. 2nd IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 451–459. 2001.

- [Pie05] Pietriga, E.: A Toolkit for Addressing HCI Issues in Visual Language Environments. In: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pp. 145–152. Dallas, TX, USA, September 2005.
- [PWK97] Pack, R. T.; Wilkes, D. M.; Kawamura, K.: A software architecture for integrated service robot development. In: *Proc. IEEE Systems, Man, and Cybernetics*, pp. 3774–3779. 1997.
- [Qui86] Quinlan, J. R.: Induction of Decision Trees. In: *Machine Learning*, volume 1:pp. 81–106, 1986.
- [Qui93] Quinlan, J. R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, USA, 1993.
- [Rap10] Rapid-I GmbH, Dortmund, Germany: *RapidMiner 4.6 - User Guide, Operator Reference, Developer Tutorial*, October 1st 2010.
- [Ras79] Rasmussen, J.: On the Structure of Knowledge - a Morphology of Mental Models in a Man- Machine System Context. Technical Report RISØ-M-2192, Risø National Laboratory, Roskilde, Denmark, 1979.
- [Ras83] Rasmussen, J.: Skills, rules, knowledge: Signals, signs, and symbols, and other distinctions in human performance models. In: *IEEE Trans. Syst., Man, Cybernet.*, volume 13:pp. 257–267, 1983.
- [Ras86] Rasmussen, J.: *Information Processing and Human-Machine Interaction - An Approach to Cognitive Engineering*. Elsevier Science Publishing, New York, NY, USA, 1986.
- [RBDF06] Ramamurthy, U.; Baars, B. J.; D'Mello, S. K.; Franklin, S.: LIDA: A Working Model of Cognition. In: *Proc. of the 7th Int. Conf. on Cognitive Modeling*, pp. 244–249. 2006.
- [Rea90] Reason, J.: *Human Error*. Cambridge University Press, Cambridge, United Kingdom, 1990.
- [RHRS07] Ritter, H.; Haschke, R.; Röthling, F.; Steil, J. J.: Manual Intelligence as a Rosetta Stone for Robot Cognition. In: *Robotics Research: The 13 International Symposium ISRR*. 2007.
- [RHW86] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.: Learning representations by back-propagating errors. In: *Nature*, volume 323:pp. 533–536, 1986.
- [RN03a] Russel, S.; Norvig, P.: *Artificial Intelligence - A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.
- [RN03b] Russel, S.; Norvig, P.: *Artificial Intelligence - A Modern Approach*, chapter Knowledge Representaion, pp. 320–374. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

- [RR83] Rouse, W.; Rouse, S.: Analysis and classification of human error. In: *IEEE Trans. Syst., Man, Cybernet.*, volume 13(4):pp. 539–549, 1983.
- [RSORS96] Rayward-Smith, V. J.; Osman, I. H.; Reeves, C. R.; Smith, G. D., editors: *Modern Heuristic Search Methods*. John Wiley & Sons, 1996.
- [RV06] Römer, P.; Visengeriyeva, L.: *db4o - schnell + kompakt*. Entwickler.Press, Frankfurt, Germany, 2006.
- [Sal85] Salzberg, S.: Heuristics for Inductive Learning. In: *Proc. of the 9th Int. Joint Conf. on Artificial Intelligence*, pp. 603–609. Los Angeles, CA, USA, 1985.
- [SB98] Sutton, R.; Barto, A.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.
- [SBF⁺96] Strube, G.; Becker, B.; Freksa, C.; Hahn, U.; Opwis, K.; Palm, G.: *Wörterbuch der Kognitionswissenschaften*. Klett-Cotta, Stuttgart, Germany, 1996.
- [Söf01a] Söffker, D.: From human-machine-interaction modeling to new concepts constructing autonomous systems: a phenomenological engineering-oriented approach. In: *Journal of Intelligent and Robotic Systems*, volume 32:pp. 191–205, 2001.
- [Söf01b] Söffker, D.: Modeling of the Knowledge-Based ‘Intelligent System’-Environment Interaction: Part I: Description and Application to Human-Machine Interaction, Part II: Systemtheoretic Aspects leading to a New Type of Autonomous Systems. In: *IEEE Int. Conf. on Systems, Man, and Cybernetics*, pp. 213–220. Tucson, AZ, USA, 2001.
- [Söf01c] Söffker, D.: *Systemtheoretic Modeling of the Knowledge-Guided Human-Machine-Interaction (In German)*. Habilitation thesis, University of Wuppertal, 2001. Published at Logos Wissenschaftsverlag, Berlin, 2003.
- [Söf04a] Söffker, D.: Understanding MMI from a system-theoretic view - Part I: Formalization MMI. In: *Proc. 9th IFAC, IFIP, IFORS, IEA Symp. Analysis, Design, and Evaluation of Human-Machine Systems*. Atlanta, Georgia, USA, 2004.
- [Söf04b] Söffker, D.: Understanding MMI from a system-theoretic view - Part II: Concepts for supervision of Human and Machine. In: *Proc. 9th IFAC, IFIP, IFORS, IEA Symp. Analysis, Design, and Evaluation of Human-Machine Systems*. Atlanta, Georgia, USA, 2004.
- [Söf08] Söffker, D.: Interaction of Intelligent and Autonomous Systems - Part I: Qualitative Structuring of Interactions. In: *MCMDS - Mathematical and Computer Modelling of Dynamical Systems*, volume 14(4):pp. 303–339, 2008.

- [Sha06] Shanahan, M.: A cognitive architecture that combines internal simulation with a global workspace. In: *Consciousness and Cognition*, volume 15:pp. 433–449, 2006.
- [SHKH03] Strube, G.; Habel, C.; Konieczny, L.; Hemforth, B.: *Handbuch der Künstlichen Intelligenz*, chapter Kognition, pp. 19–72. Oldenbourg Wissenschaftsverlag, Munich, Germany, 4th edition, 2003.
- [Sim83] Simon, H. A.: *Machine Learning: An Artificial Intelligence Approach*, chapter Why Should Machines Learn?, pp. 25–37. Morgan Kaufmann Publishers, Los Altos, CA, USA, 1983.
- [Sin05] Singh, P.: *EM-ONE: An Architecture for Reflective Commonsense Thinking*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June 2005.
- [Slo01] Sloman, A.: Varieties of Affect and the CogAff Architecture Schema. In: *Symposium on Emotion, Cognition, and Affective Computing AISB’01 Convention*. 2001.
- [SM64] Sonquist, J. A.; Morgan, J. N.: The detection of interaction effects: A report on a computer program for the selection of optimal combinations of explanatory variables. Monograph 35, University of Michigan, Survey Research Center, Institute for Social Research, Ann Arbor, 1964.
- [SMP01] Sun, R.; Merrill, E.; Peterson, T.: From implicit skills to explicit knowledge: A bottom-up model of skill learning. In: *Cognitive Science*, volume 25(2):pp. 203–244, 2001.
- [SNW⁺04] Skubic, M.; Noelle, D.; Wilkes, M.; Kawamura, K.; Keller, J. M.: A biologically inspired adaptive working memory for robots. In: *AAAI Fall Symposium, Workshop on The Intersection of Cognitive Science and Robotics: From Interfaces to Intelligence*. Washington, D.C., October 21st - 24th 2004.
- [Sow92] Sowa, J. F.: *Encyclopedia of Artificial Intelligence*, chapter Semantic Networks. John Wiley & Sons, New York, NY, USA, 2nd edition, 1992.
- [Sow00] Sowa, J. F.: *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., Pacific Grove, CA, USA, 2000.
- [SPS⁺04] Sofge, D.; Perzanowski, D.; Skubic, M.; Bugajska, M.; Trafton, J. G.; Casimatis, N.; Brock, D.; Adams, W.; Schultz, A.: Cognitive Tools for Humanoid Robotics in Space. In: *Proc. of the 16th IFAC Symp. on Automatic Control in Aerospace*. 2004.
- [SS02] Schölkopf, B.; Smola, A. J.: *Learning with Kernels*. MIT Press, Cambridge, MA, USA, 2002.

-
- [Sun07] Sun, R.: *Integrated Models of Cognitive Systems*, chapter The Motivational and Metacognitive Control in CLARION, pp. 63–75. Oxford University Press, New York, NY, USA, 2007.
 - [SVS99] Stamou, G.; Vogiatzis, D.; Stove, S.: Bridging the gap between subsymbolic and symbolic techniques: A pragmatic approach. In: *CSCC'99, Circuits Systems Communications and Computers*. Athens, Greece, 1999.
 - [SZ06] Sun, R.; Zhang, X.: Accounting for a variety of reasoning data within a cognitive architecture. In: *Journal of Experimental and Theoretical Artificial Intelligence*, volume 18(2):pp. 169–191, 2006.
 - [TBF05] Thrun, S.; Burgard, W.; Fox, D.: *Probotics*. The MIT Press, Cambridge, MA, USA, 2005.
 - [TD03] Taatgen, N. A.; Dijkstra, M.: Constraints on Generalization: Why are past-tense irregularization errors so rare? In: *Proc. of the 25th annual conference of the cognitive science society*, pp. 1146–1151. Mahwah, NJ, USA, 2003.
 - [TD07] Tribelhorn, B.; Dodds, Z.: Evaluating the Roomba: A low-cost, ubiquitous platform for robotics research and education. In: *2007 IEEE Int. Conf. on Robotics and Automation*, pp. 1393–1399. 2007.
 - [Thu38] Thurstone, L. L.: *Primary mental abilities*. University of Chicago Press, Chicago, IL, USA, 1938.
 - [TK08] Theodoridis, S.; Koutroumbas, K.: *Pattern Recognition*. Academic Press, Burlington/San Diego/London, 4th edition, 2008.
 - [UAB⁺08] Urmson, C.; Anhalt, J.; Bae, H.; Bagnell, J. A.; Baker, C.; Bittner, R. E.; Brown, T.; Clark, M. N.; Darms, M.; Demitrish, D.; Dolan, J.; Duggins, D.; Ferguson, D.; Galatali, T.; Geyer, C. M.; Gittleman, M.; Harbaugh, S.; Hebert, M.; Howard, T.; Kolski, S.; Likhachev, M.; Litkouhi, B.; Kelly, A.; McNaughton, M.; Miller, N.; Nickolaou, J.; Peterson, K.; Pilnick, B.; Rajkumar, R.; Rybski, P.; Sadekar, V.; Salesky, B.; Seo, Y.-W.; Singh, S.; Snider, J. M.; Struble, J. C.; Stentz, A.; Taylor, M.; Whittaker, W. L.; Wolkowicki, Z.; Zhang, W.; Ziglar, J.: Autonomous driving in urban environments: Boss and the Urban Challenge. In: *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, volume 25(8):pp. 425–466., June 2008.
 - [Val98] Valk, R.: Petri nets as token objects: An introduction to elementary object nets. In: Desel, J.; Silva, M., editors, *Application and Theory of Petri Nets*, number 1420 in LNCS, pp. 1–25. ICATPN '98, Springer, Lisbon, June 1998.
 - [VCP⁺95] Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; Blythe, J.: Integrating Planning and Learning: The PRODIGY Architecture. In: *Journal*

- of Experimental and Theoretical Artificial Intelligence*, volume 7(1):pp. 81–120, 1995.
- [Ver10] VersaLogic Corporation, Eugene, OR, USA: *EBX-12 Reference Manual*, Revision 6, November 2010.
- [vLMV⁺03] v. Lüde, R.; Moldt, D.; Valk, R.; Köhler, M.; Langer, R.; Rölke, H.; Spresny, D.: *Sozionik - Modellierung soziologischer Theorie*. Lit Verlag, Münster, 2003.
- [VMS07] Vernon, D.; Metta, G.; Sandini, G.: A Survey of Artificial Cognitive Systems: Implications for the Autonomous Development of Mental Capabilities in Computational Agents. In: *IEEE Transactions on Evolutionary Computation*, volume 11(2):pp. 151–180, 2007.
- [Wal01] Wallich, P.: Mindstorms: not just a kid’s toy. In: *IEEE Spectrum*, volume 38(9):pp. 52 – 57, 2001.
- [Wat89] Watkins, C. J. C. H.: *Learning from Delayed Rewards*. PhD thesis, King’s College, Cambridge University, Cambridge, GB, 1989.
- [WD92] Watkins, C. J. C. H.; Dayan, P.: Q-learning. In: *Machine Learning*, volume 8:pp. 279–292, 1992.
- [Wen04] Weng, J.: Developmental Robotics: Theory and Experiments. In: *Int. Journal of Humanoid Robotics*, volume 1(2):pp. 199–236, 2004.
- [Wer06] Werther, B.: *Kognitive Modellierung mit Farbigen Petrinetzen zur Analyse menschlichen Verhaltens*. Dr.-Ing. thesis, Fakultät für Maschinenbau der Technischen Universität Carolo-Wilhelmina zu Braunschweig, Braunschweig, 2006.
- [WF05] Witten, I. H.; Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2nd edition, 2005.
- [WKG⁺07] Wersing, H.; Kirstein, S.; Götting, M.; Brandl, H.; Dunn, M.; Mikhailova, I.; Goerick, C.; Steil, J. J.; Ritter, H.; Körner, E.: Online Learning of Objects and Faces in an Integrated Biologically Motivated Architecture. In: *Int. Conf. in Computer Vision Systems*. Bielefeld, Germany, 2007.
- [Zad96] Zadeh, L. A.: *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers by Lotfi A. Zadeh*. World Scientific Publishing Co., Singapore, 1996.

Im Rahmen von Forschungs- und Projektarbeiten im Lehrstuhl SRS wurden von Dr.-Ing. Dennis Gamrad und Univ.-Prof. Dr.-Ing. Dirk Söffker die nachstehenden Studien-, Diplom-, Bachelor-, Projekt- und Masterarbeiten inhaltlich betreut, wobei Bestandteile und Ergebnisse aus den Forschungs- und Projektarbeiten sowie den studentischen Qualifikationsarbeiten wechselseitig in die jeweiligen Arbeiten und somit auch in diese Promotionsarbeit eingeflossen sind.

- [Bou08] Boussairi, H.: *Petri-Netz-basierte Implementierung eines verfahrenstechnischen Prozessmodells - SOM-basierte automatische Überwachung der Mensch-Maschine-Interaktion*. Diploma thesis, Department of Computational and Cognitive Sciences, Faculty of Engineering Sciences, University of Duisburg-Essen, Duisburg, Germany, August 2008.
- [Fu08] Fu, X.: *Analysis and Modeling of a Driver-Vehicle-Interaction*. Master thesis, Chair of Dynamics and Control (SRS), Department of Mechanical Engineering, Faculty of Engineering Sciences, University of Duisburg-Essen, Duisburg, Germany, December 2008.
- [Has08] Hasselberg, A.: *Erweiterung einer Petri-Netz-basierten Simulationsumgebung zur automatisierten Erfassung menschlichen Fehlverhaltens*. Studienarbeit, Chair of Dynamics and Control (SRS), Department of Mechanical Engineering, Faculty of Engineering Sciences, University of Duisburg-Essen, Duisburg, Germany, December 2008.
- [HH08] Hasselberg, A.; Hitzbleck, B.: *Konzeption und Aufbau eines mobilen kognitiven Roboters für interaktive Aufgaben*. Projektarbeit, Chair of Dynamics and Control (SRS), Department of Mechanical Engineering, Faculty of Engineering Sciences, University of Duisburg-Essen, Duisburg, Germany, July 2008.
- [Lei08] Lei, D.: *User Guided and Autonomous Navigation of a Mobile Robot*. Bachelor thesis, Chair of Dynamics and Control (SRS), Department of Mechanical Engineering, Faculty of Engineering Sciences, University of Duisburg-Essen, Duisburg, Germany, December 2008.
- [Oug08] Oughalmi, A.: *Aufbau der Schnittstelle und Inbetriebnahme eines verteilten Systems*. Projektarbeit, Chair of Dynamics and Control (SRS), Department of Mechanical Engineering, Faculty of Engineering Sciences, University of Duisburg-Essen, Duisburg, Germany, February 2008.
- [Yan10] Yang, Z.: *Implementation of a RFID System to a Mobile Robot and Communication using Middleware Software*. Master thesis, Chair of Dynamics and Control (SRS), Department of Mechanical Engineering, Faculty of Engineering Sciences, University of Duisburg-Essen, Duisburg, Germany, October 2010.